

The code of the package `nicematrix`*

F. Pantigny
fpantigny@wanadoo.fr

October 12, 2024

Abstract

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

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

1 Declaration of the package and packages loaded

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

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

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

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

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

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

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

```
14 \RequirePackage { amsmath }

15 \RequirePackage { array }
```

*This document corresponds to the version 6.28c of `nicematrix`, at the date of 2024/08/22.

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

```

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

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

```

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

```

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

```

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

```

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

```

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

```

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

```

```

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

```

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

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

2 Security test

Within the package `nicematrix`, we will have to test whether a cell of a `{NiceTabular}` is empty. For the cells of the columns of type `p`, `b`, `m`, `X` and `V`, we will test whether the cell is syntactically empty (that is to say that there is only spaces between the ampersands `&`). That test will be done with the command `\@@_test_if_empty:` by testing if the two first tokens in the cells are (during the TeX process) are `\ignorespaces` and `\unskip`.

However, if, one day, there is a changement in the implementation of `array`, maybe that this test will be broken (and `nicematrix` also).

That's why, by security, we will take a test in a small `{tabular}` composed in the box `\l_tmpa_box` used as sandbox.

```
64 \@@_msg_new:nn { Internal~error }
65 {
66   Potential~problem~when~using~nicematrix.\\
67   The~package~nicematrix~have~detected~a~modification~of~the~
68   standard~environment~{array}~(of~the~package~array).~Maybe~you~will~encounter~
69   some~slight~problems~when~using~nicematrix.~If~you~don't~want~to~see~
70   this~message~again,~load~nicematrix~with:~\token_to_str:N
71   \usepackage[no-test-for-array]{nicematrix}.
72 }
```

```
73 \@@_msg_new:nn { mdwtab~loaded }
74 {
75   The~packages~'mdwtab'~and~'nicematrix'~are~incompatible.~
76   This~error~is~fatal.
77 }
```

```
78 \cs_new_protected:Npn \@@_security_test:n #1
79 {
80   \peek_meaning:NTF \ignorespaces
81   { \@@_security_test_i:w }
82   { \@@_error:n { Internal~error } }
83   #1
84 }
```

```
85 \bool_if:NTF \c_@@_tagging_array_bool
86 {
87   \cs_new_protected:Npn \@@_security_test_i:w \ignorespaces #1
88   {
89     \peek_meaning:NF \textonly@unskip { \@@_error:n { Internal~error } }
90     #1
91   }
92 }
93 {
94   \cs_new_protected:Npn \@@_security_test_i:w \ignorespaces #1
95   {
96     \peek_meaning:NF \unskip { \@@_error:n { Internal~error } }
97     #1
98   }
99 }
```

Here, the box `\l_tmpa_box` will be used as sandbox to take our security test.

```

100 \hook_gput_code:nnn { begindocument / end } { . }
101 {
102   \IfPackageLoadedTF { mdwtab }
103     { \@@_fatal:n { mdwtab~loaded } }
104     {
105       \bool_if:NF \g_@@_no_test_for_array_bool
106       {
107         \group_begin:
108         \hbox_set:Nn \l_tmpa_box
109           {
110             \begin { tabular } { c > { \@@_security_test:n } c c }
111             text & & text
112             \end { tabular }
113           }
114         \group_end:
115       }
116     }
117 }

```

3 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`).

```

118 \cs_new_protected:Npn \@@_collect_options:n #1
119 {
120   \peek_meaning:NTF [
121     { \@@_collect_options:nw { #1 } }
122     { #1 { } }
123 }

```

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

```

124 \NewDocumentCommand \@@_collect_options:nw { m r[] }
125 { \@@_collect_options:nn { #1 } { #2 } }
126
127 \cs_new_protected:Npn \@@_collect_options:nn #1 #2
128 {
129   \peek_meaning:NTF [
130     { \@@_collect_options:nnw { #1 } { #2 } }
131     { #1 { #2 } }
132 }
133
134 \cs_new_protected:Npn \@@_collect_options:nnw #1#2[#3]
135 { \@@_collect_options:nn { #1 } { #2 , #3 } }

```

4 Technical definitions

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

```

136 \tl_const:Nn \c_@@_b_tl { b }
137 \tl_const:Nn \c_@@_c_tl { c }
138 \tl_const:Nn \c_@@_l_tl { l }
139 \tl_const:Nn \c_@@_r_tl { r }
140 \tl_const:Nn \c_@@_all_tl { all }
141 \tl_const:Nn \c_@@_dot_tl { . }
142 \tl_const:Nn \c_@@_default_tl { default }
143 \tl_const:Nn \c_@@_star_tl { * }
144 \str_const:Nn \c_@@_star_str { * }
145 \str_const:Nn \c_@@_r_str { r }
146 \str_const:Nn \c_@@_c_str { c }
147 \str_const:Nn \c_@@_l_str { l }
148 \str_const:Nn \c_@@_R_str { R }
149 \str_const:Nn \c_@@_C_str { C }
150 \str_const:Nn \c_@@_L_str { L }
151 \str_const:Nn \c_@@_j_str { j }
152 \str_const:Nn \c_@@_si_str { si }

```

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).

```

153 \tl_new:N \l_@@_argspec_tl

154 \cs_generate_variant:Nn \seq_set_split:Nnn { N o n }
155 \cs_generate_variant:Nn \str_lowercase:n { o }
156 \cs_generate_variant:Nn \tl_build_put_right:Nn { N o }
157 \prg_generate_conditional_variant:Nnn \clist_if_in:Nn { N e } { T , TF }
158 \prg_generate_conditional_variant:Nnn \tl_if_empty:n { e } { T }
159 \prg_generate_conditional_variant:Nnn \tl_if_head_eq_meaning:nN { o N } { TF }
160 \cs_generate_variant:Nn \dim_min:nn { v n }
161 \cs_generate_variant:Nn \dim_max:nn { v n }

162 \hook_gput_code:nnn { begindocument } { . }
163 {
164   \IfPackageLoadedTF { tikz }
165   {

```

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.

```

166 \tl_const:Nn \c_@@_pgfortikzpicture_tl { \exp_not:N \tikzpicture }
167 \tl_const:Nn \c_@@_endpgfortikzpicture_tl { \exp_not:N \endtikzpicture }
168 }
169 {
170 \tl_const:Nn \c_@@_pgfortikzpicture_tl { \exp_not:N \pgfpicture }
171 \tl_const:Nn \c_@@_endpgfortikzpicture_tl { \exp_not:N \endpgfpicture }
172 }
173 }

```

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`).

```

174 \IfClassLoadedTF { revtex4-1 }
175 { \bool_const:Nn \c_@@_revtex_bool \c_true_bool }

```

```

176 {
177   \IfClassLoadedTF { revtex4-2 }
178   { \bool_const:Nn \c_@@_revtex_bool \c_true_bool }
179   {

```

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

```

180   \cs_if_exist:NT \rvtx@ifformat@geq
181   { \bool_const:Nn \c_@@_revtex_bool \c_true_bool }
182   { \bool_const:Nn \c_@@_revtex_bool \c_false_bool }
183 }
184 }

```

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.

```

185 \cs_new_protected:Npn \@@_provide_pgfsyspdfmark:
186 {
187   \iow_now:Nn \@mainaux
188   {
189     \ExplSyntaxOn
190     \cs_if_free:NT \pgfsyspdfmark
191     { \cs_set_eq:NN \pgfsyspdfmark \@gobblethree }
192     \ExplSyntaxOff
193   }
194   \cs_gset_eq:NN \@@_provide_pgfsyspdfmark: \prg_do_nothing:
195 }

```

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.

```

196 \ProvideDocumentCommand \iddots { }
197 {
198   \mathinner
199   {
200     \tex_mkern:D 1 mu
201     \box_move_up:nn { 1 pt } { \hbox { . } }
202     \tex_mkern:D 2 mu
203     \box_move_up:nn { 4 pt } { \hbox { . } }
204     \tex_mkern:D 2 mu
205     \box_move_up:nn { 7 pt }
206     { \vbox:n { \kern 7 pt \hbox { . } } }
207     \tex_mkern:D 1 mu
208   }
209 }

```

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.

```

210 \hook_gput_code:nnn { begindocument } { . }
211 {
212   \IfPackageLoadedT { booktabs }
213   { \iow_now:Nn \@mainaux \nicematrix@redefine@check@rerun }
214 }
215 \cs_set_protected:Npn \nicematrix@redefine@check@rerun
216 {
217   \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`).

```

218 \cs_set_protected:Npn \pgfutil@check@rerun ##1 ##2
219 {
220   \str_if_eq:eeF { nm- } { \tl_range:nnn { ##1 } 1 3 }
221   { \@@_old_pgfutil@check@rerun { ##1 } { ##2 } }
222 }
223 }

```

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

```

224 \hook_gput_code:nnn { begindocument } { . }
225 {
226   \IfPackageLoadedF { colortbl }
227   {

```

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.

```

228   \cs_set_protected:Npn \CT@arc@ { }
229   \cs_set_nopar:Npn \arrayrulecolor #1 # { \CT@arc@ { #1 } }
230   \cs_set_nopar:Npn \CT@arc@ #1 #2
231   {
232     \dim_compare:nNnT \baselineskip = \c_zero_dim \noalign
233     { \cs_gset_nopar:Npn \CT@arc@ { \color #1 { #2 } } }
234   }

```

Idem for `\CT@drs@`.

```

235   \cs_set_nopar:Npn \doublerulesepcolor #1 # { \CT@drs@ { #1 } }
236   \cs_set_nopar:Npn \CT@drs@ #1 #2
237   {
238     \dim_compare:nNnT \baselineskip = \c_zero_dim \noalign
239     { \cs_gset:Npn \CT@drsc@ { \color #1 { #2 } } }
240   }
241   \cs_set_nopar:Npn \hline
242   {
243     \noalign { \ifnum 0 = ` } \fi
244     \cs_set_eq:NN \hskip \vskip
245     \cs_set_eq:NN \vrule \hrule
246     \cs_set_eq:NN \@width \@height
247     { \CT@arc@ \vline }
248     \futurelet \reserved@a
249     \@xhline
250   }
251 }
252 }

```

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.

```

253 \cs_set_nopar:Npn \@@_standard_cline #1 { \@@_standard_cline:w #1 \q_stop }
254 \cs_set_nopar:Npn \@@_standard_cline:w #1-#2 \q_stop
255 {
256   \int_if_zero:nT \l_@@_first_col_int { \omit & }
257   \int_compare:nNnT { #1 } > \c_one_int
258   { \multispan { \int_eval:n { #1 - 1 } } & }
259   \multispan { \int_eval:n { #2 - #1 + 1 } }
260   {
261     \CT@arc@
262     \leaders \hrule \@height \arrayrulewidth \hfill

```

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

¹See question 99041 on TeX StackExchange.

```

263     \skip_horizontal:N \c_zero_dim
264 }

```

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`.

```

265 \everycr { }
266 \cr
267 \noalign { \skip_vertical:N -\arrayrulewidth }
268 }

```

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.

```

269 \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`.

```

270 { \@@_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*.

```

271 \cs_set:Npn \@@_cline_i:nn #1 #2 { \@@_cline_i:w #1|#2- \q_stop }
272 \cs_generate_variant:Nn \@@_cline_i:nn { e n }
273 \cs_set:Npn \@@_cline_i:w #1|#2-#3 \q_stop
274 {
275     \tl_if_empty:nTF { #3 }
276     { \@@_cline_iii:w #1|#2-#2 \q_stop }
277     { \@@_cline_ii:w #1|#2-#3 \q_stop }
278 }
279 \cs_set:Npn \@@_cline_ii:w #1|#2-#3-\q_stop
280 { \@@_cline_iii:w #1|#2-#3 \q_stop }
281 \cs_set:Npn \@@_cline_iii:w #1|#2-#3 \q_stop
282 {

```

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).

```

283     \int_compare:nNnT { #1 } < { #2 }
284     { \multispan { \int_eval:n { #2 - #1 } } & }
285     \multispan { \int_eval:n { #3 - #2 + 1 } }
286     {
287         \CT@arc@
288         \leaders \hrule \@height \arrayrulewidth \hfill
289         \skip_horizontal:N \c_zero_dim
290     }

```

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

```

291 \peek_meaning_remove_ignore_spaces:NTF \cline
292 { & \@@_cline_i:en { \int_eval:n { #3 + 1 } } }
293 { \everycr { } \cr }
294 }

```

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

```

295 \cs_set_eq:NN \@@_math_toggle: \c_math_toggle_token

```

```

296 \cs_generate_variant:Nn \@@_set_CT@arc@:n { o }
297 \cs_new_protected:Npn \@@_set_CT@arc@:n #1
298 {
299     \tl_if_blank:nF { #1 }
300     {
301         \tl_if_head_eq_meaning:nNTF { #1 } [
302             { \cs_set_nopar:Npn \CT@arc@ { \color #1 } }

```



```

303         { \cs_set_nopar:Npn \CT@arc@ { \color { #1 } } }
304     }
305 }

306 \cs_generate_variant:Nn \@@_set_CT@drsc@:n { o }
307 \cs_new_protected:Npn \@@_set_CT@drsc@:n #1
308 {
309     \tl_if_head_eq_meaning:nNTF { #1 } [
310         { \cs_set_nopar:Npn \CT@drsc@ { \color #1 } }
311         { \cs_set_nopar:Npn \CT@drsc@ { \color { #1 } } }
312     ]

```

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

```

313 \cs_generate_variant:Nn \@@_exp_color_arg:Nn { N o }
314 \cs_new:Npn \@@_exp_color_arg:Nn #1 #2
315 {
316     \tl_if_head_eq_meaning:nNTF { #2 } [
317         { #1 #2 }
318         { #1 { #2 } }
319     ]

```

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

```

320 \cs_generate_variant:Nn \@@_color:n { o }
321 \cs_new_protected:Npn \@@_color:n #1
322 { \tl_if_blank:nF { #1 } { \@@_exp_color_arg:Nn \color { #1 } } }

323 \cs_new_protected:Npn \@@_rescan_for_spanish:N #1
324 {
325     \tl_set_rescan:Nno
326     #1
327     {
328         \char_set_catcode_other:N >
329         \char_set_catcode_other:N <
330     }
331     #1
332 }

```

5 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.

```

333 \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).

```

334 \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.

```

335 \NewExpandableDocumentCommand \NiceMatrixLastEnv { }
336 { \int_use:N \g_@@_env_int }

```

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

```
337 \cs_new_protected:Npn \@@_qpoint:n #1
338 { \pgfpointanchor { \@@_env: - #1 } { center } }
```

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

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

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

```
340 \bool_new:N \g_@@_delims_bool
341 \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.

```
342 \bool_new:N \l_@@_preamble_bool
343 \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.

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

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

```
345 \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.

```
346 \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).

```
347 \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.).

```
348 \dim_new:N \l_@@_col_width_dim
349 \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.

```
350 \int_new:N \g_@@_row_total_int
351 \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).

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

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

```
353 \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.

```
354 \tl_new:N \l_@@_hpos_cell_tl
355 \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`.

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

Idem for the mono-row blocks.

```
357 \dim_new:N \g_@@_blocks_ht_dim
358 \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}`).

```
359 \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`.

```
360 \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.

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

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

```
362 \bool_new:N \l_@@_notes_detect_duplicates_bool
363 \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.

```
364 \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).

```
365 \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).

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

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

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

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

```
368 \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.

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

```
370 \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 }`).

```
371 \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.

```
372 \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.

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

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

```
375 \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}`).

```
376 \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`).

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

For `\multicolumn`.

```
378 \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`.

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

```
380 \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 `:`.

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

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

```
383 \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`).

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

```
385 \cs_new_protected:Npn \@@_test_if_math_mode:
```

```
386 {
```

```
387   \if_mode_math: \else:
```

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

```
389   \fi:
```

```
390 }
```

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.

```
391 \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”.

```
392 \colorlet { nicematrix-last-col } { . }
393 \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`).

```
394 \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*.

```
395 \tl_new:N \g_@@_com_or_env_str
396 \tl_gset:Nn \g_@@_com_or_env_str { environment }
```

```
397 \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:onTF` and not `\tl_if_eq:NnTF` because we need to be fully expandable).

```
398 \cs_new:Npn \@@_full_name_env:
399 {
400   \str_if_eq:onTF \g_@@_com_or_env_str { command }
401     { command \space \c_backslash_str \g_@@_name_env_str }
402     { environment \space \{ \g_@@_name_env_str \} }
403 }
```

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

```
404 \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.

```
405 \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.

```
406 \tl_new:N \g_@@_pre_code_before_tl
407 \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.

```
408 \tl_new:N \g_@@_pre_code_after_tl
409 \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.

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

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

```
411 \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.

```
412 \int_new:N \l_@@_old_iRow_int
413 \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).

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

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

```
415 \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.

```
416 \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.

```
417 \bool_new:N \l_@@_X_columns_aux_bool
418 \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.

```
419 \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).

```
420 \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).

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

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

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

```
422 \tl_new:N \l_@@_code_before_tl
423 \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).

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

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

```

425 \dim_new:N \l_@@_x_initial_dim
426 \dim_new:N \l_@@_y_initial_dim
427 \dim_new:N \l_@@_x_final_dim
428 \dim_new:N \l_@@_y_final_dim

```

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

```

429 \dim_new:N \l_@@_tmpc_dim
430 \dim_new:N \l_@@_tmpd_dim

431 \dim_new:N \g_@@_dp_row_zero_dim
432 \dim_new:N \g_@@_ht_row_zero_dim
433 \dim_new:N \g_@@_ht_row_one_dim
434 \dim_new:N \g_@@_dp_ante_last_row_dim
435 \dim_new:N \g_@@_ht_last_row_dim
436 \dim_new:N \g_@@_dp_last_row_dim

```

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

```

437 \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”.

```

438 \dim_new:N \g_@@_width_last_col_dim
439 \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.

```

440 \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!).

```

441 \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}`.

```

442 \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.

```

443 \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 sequence.

```

444 \seq_new:N \l_@@_corners_cells_seq

```

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`).

```
445 \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.

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

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

```
447 \seq_new:N \g_@@_multicolumn_cells_seq
448 \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`).

```
449 \int_new:N \l_@@_row_min_int
450 \int_new:N \l_@@_row_max_int
451 \int_new:N \l_@@_col_min_int
452 \int_new:N \l_@@_col_max_int
```

The following counters will be used when drawing the rules.

```
453 \int_new:N \l_@@_start_int
454 \int_set_eq:NN \l_@@_start_int \c_one_int
455 \int_new:N \l_@@_end_int
456 \int_new:N \l_@@_local_start_int
457 \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.

```
458 \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).

```
459 \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`.

```
460 \tl_new:N \l_@@_fill_tl
461 \tl_new:N \l_@@_opacity_tl
462 \tl_new:N \l_@@_draw_tl
463 \seq_new:N \l_@@_tikz_seq
464 \clist_new:N \l_@@_borders_clist
465 \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.

```
466 \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`.

```
467 \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.

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

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

```
469 \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`).

```
470 \str_new:N \l_@@_hpos_block_str
471 \str_set:Nn \l_@@_hpos_block_str { c }
472 \bool_new:N \l_@@_hpos_of_block_cap_bool
473 \bool_new:N \l_@@_p_block_bool
```

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

```
474 \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).

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

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

```
476 \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).

```
477 \bool_new:N \l_@@_vlines_block_bool
478 \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.

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

```
480 \dim_new:N \l_@@_submatrix_extra_height_dim
481 \dim_new:N \l_@@_submatrix_left_xshift_dim
482 \dim_new:N \l_@@_submatrix_right_xshift_dim
483 \clist_new:N \l_@@_hlines_clist
484 \clist_new:N \l_@@_vlines_clist
485 \clist_new:N \l_@@_submatrix_hlines_clist
486 \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}`).

```
487 \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.

```
488 \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`).

```
489 \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.

```
490 \int_new:N \l_@@_first_row_int
491 \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.

```
492 \int_new:N \l_@@_first_col_int
493 \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).

```
494 \int_new:N \l_@@_last_row_int
495 \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

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

Idem for `\l_@@_last_col_without_value_bool`

```
497 \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.

```
498 \int_new:N \l_@@_last_col_int
499 \int_set:Nn \l_@@_last_col_int { -2 }
```

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

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

```

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

```

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

```

500 \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`).

```

501 \bool_new:N \l_@@_in_last_col_bool

```

Some utilities

```

502 \cs_set_protected:Npn \@@_cut_on_hyphen:w #1-#2\q_stop
503 {
504   \cs_set_nopar:Npn \l_tmpa_tl { #1 }
505   \cs_set_nopar:Npn \l_tmpb_tl { #2 }
506 }

```

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.

```

507 \cs_new_protected:Npn \@@_expand_clist:N #1
508 {
509   \clist_if_in:NVF #1 \c_@@_all_tl
510   {
511     \clist_clear:N \l_tmpa_clist
512     \clist_map_inline:Nn #1
513     {
514       \tl_if_in:nnTF { ##1 } { - }
515       { \@@_cut_on_hyphen:w ##1 \q_stop }
516       {
517         \cs_set_nopar:Npn \l_tmpa_tl { ##1 }
518         \cs_set_nopar:Npn \l_tmpb_tl { ##1 }
519       }
520       \int_step_inline:nnn \l_tmpa_tl \l_tmpb_tl
521       { \clist_put_right:Nn \l_tmpa_clist { ####1 } }
522     }
523     \tl_set_eq:NN #1 \l_tmpa_clist
524   }
525 }

```

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.

```

526 \hook_gput_code:nnn { begindocument } { . }
527 {
528   \dim_const:Nn \c_@@_shift_Ldots_last_row_dim { 0.5 em }
529   \dim_const:Nn \c_@@_shift_exterior_Vdots_dim { 0.6 em }
530   \dim_const:Nn \c_@@_innersep_middle_dim { 0.17 em }
531 }

```

6 The command `\tablarnote`

Of course, it's possible to use `\tablarnote` 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 `\tablarnote` in that `\caption` makes sens only if the `\caption` is *before* the `{\tabular}`.
- It's also possible to use `\tablarnote` 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 tablarnote}`. The first component is the optional argument (between square brackets) of the command `\tablarnote` (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 `tablarnote` 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.

```
532 \newcounter { tablarnote }
```

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_@@_tablarnote_int`.

```
533 \int_new:N \g_@@_tablarnote_int
534 \cs_set:Npn \theHtablarnote { \int_use:N \g_@@_tablarnote_int }
535 \seq_new:N \g_@@_notes_seq
536 \seq_new:N \g_@@_notes_in_caption_seq
```

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

```
537 \tl_new:N \g_@@_tablarnote_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.

```
538 \seq_new:N \l_@@_notes_labels_seq
539 \newcounter{nicematrix_draft}
```

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

```

540 \cs_new_protected:Npn \@@_notes_format:n #1
541 {
542   \setcounter { nicematrix_draft } { #1 }
543   \@@_notes_style:n { nicematrix_draft }
544 }

```

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

```

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

```

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

```

546 \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`.

```

547 \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`.

```

548 \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`).

```

549 \hook_gput_code:nnn { begindocument } { . }
550 {
551   \IfPackageLoadedTF { enumitem }
552   {

```

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.

```

553     \newlist { tabularnotes } { enumerate } { 1 }
554     \setlist [ tabularnotes ]
555     {
556       topsep = 0pt ,
557       noitemsep ,
558       leftmargin = * ,
559       align = left ,
560       labelsep = 0pt ,
561       label =
562         \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotesi } } ,
563     }
564     \newlist { tabularnotes* } { enumerate* } { 1 }
565     \setlist [ tabularnotes* ]
566     {
567       afterlabel = \nobreak ,
568       itemjoin = \quad ,
569       label =
570         \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotes*i } }
571     }

```

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`.

```

572     \NewDocumentCommand \tabularnote { o m }
573     {
574       \bool_lazy_or:nnT { \cs_if_exist_p:N \capttype } \l_@@_in_env_bool

```

```

575         {
576         \bool_lazy_and:nnTF { ! \l_@@_tabular_bool } \l_@@_in_env_bool
577         { \@@_error:n { tabularnote~forbidden } }
578         {
579         \bool_if:NTF \l_@@_in_caption_bool
580         \@@_tabularnote_caption:nn
581         \@@_tabularnote:nn
582         { #1 } { #2 }
583         }
584     }
585 }
586 }
587 {
588     \NewDocumentCommand \tabularnote { o m }
589     {
590         \@@_error_or_warning:n { enumitem~not~loaded }
591         \@@_gredirect_none:n { enumitem~not~loaded }
592     }
593 }
594 }
595 \cs_new_protected:Npn \@@_test_first_novalue:nnn #1 #2 #3
596 { \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`.

```

597 \cs_new_protected:Npn \@@_tabularnote:nn #1 #2
598 {

```

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).

```

599     \int_zero:N \l_tmpa_int
600     \bool_if:NT \l_@@_notes_detect_duplicates_bool
601     {

```

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`.

```

602     \int_zero:N \l_tmpb_int
603     \seq_map_indexed_inline:Nn \g_@@_notes_seq
604     {
605         \@@_test_first_novalue:nnn ##2 { \int_incr:N \l_tmpb_int }
606         \tl_if_eq:nnT { { #1 } { #2 } } { ##2 }
607         {
608             \tl_if_novalue:nTF { #1 }
609             { \int_set_eq:NN \l_tmpa_int \l_tmpb_int }
610             { \int_set:Nn \l_tmpa_int { ##1 } }
611             \seq_map_break:
612         }
613     }
614     \int_if_zero:nF \l_tmpa_int
615     { \int_add:Nn \l_tmpa_int \g_@@_notes_caption_int }
616 }
617 \int_if_zero:nT \l_tmpa_int
618 {

```

```

619     \seq_gput_right:Nn \g_@@_notes_seq { { #1 } { #2 } }
620     \tl_if_novalue:nT { #1 } { \int_gincr:N \c@tabularnote }
621   }
622   \seq_put_right:Ne \l_@@_notes_labels_seq
623   {
624     \tl_if_novalue:nTF { #1 }
625     {
626       \@@_notes_format:n
627       {
628         \int_eval:n
629         {
630           \int_if_zero:nTF \l_tmpa_int
631             \c@tabularnote
632             \l_tmpa_int
633         }
634       }
635     }
636     { #1 }
637   }
638   \peek_meaning:NF \tabularnote
639   {

```

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`.

```

640     \hbox_set:Nn \l_tmpa_box
641     {

```

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

```

642     \@@_notes_label_in_tabular:n
643     {
644       \seq_use:Nnnn
645       \l_@@_notes_labels_seq { , } { , } { , }
646     }
647   }

```

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

```

648     \int_gdecr:N \c@tabularnote
649     \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.

```

650     \int_gincr:N \g_@@_tabularnote_int
651     \refstepcounter { tabularnote }
652     \int_compare:nNnT \l_tmpa_int = \c@tabularnote
653     { \int_gincr:N \c@tabularnote }
654     \seq_clear:N \l_@@_notes_labels_seq
655     \bool_lazy_or:nnTF
656     { \tl_if_eq_p:NN \l_@@_hpos_cell_tl \c_@@_c_tl }
657     { \tl_if_eq_p:NN \l_@@_hpos_cell_tl \c_@@_r_tl }
658     {
659       \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).

```

660     \skip_horizontal:n { \box_wd:N \l_tmpa_box }
661   }
662   { \box_use:N \l_tmpa_box }
663 }
664 }

```

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`.

```

665 \cs_new_protected:Npn \@@_tabularnote_caption:nn #1 #2
666 {
667   \bool_if:NTF \g_@@_caption_finished_bool
668   {
669     \int_compare:nNtT \c@tabularnote = \g_@@_notes_caption_int
670     { \int_gzero:N \c@tabularnote }

```

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

```

671     \seq_if_in:NnF \g_@@_notes_in_caption_seq { { #1 } { #2 } }
672     { \@@_error:n { Identical-notes-in-caption } }
673   }
674 {

```

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

```

675     \seq_if_in:NnTF \g_@@_notes_in_caption_seq { { #1 } { #2 } }
676     {

```

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

```

677         \bool_gset_true:N \g_@@_caption_finished_bool
678         \int_gset_eq:NN \g_@@_notes_caption_int \c@tabularnote
679         \int_gzero:N \c@tabularnote
680     }
681     { \seq_gput_right:Nn \g_@@_notes_in_caption_seq { { #1 } { #2 } } }
682 }

```

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!

```

683     \tl_if_novalue:nT { #1 } { \int_gincr:N \c@tabularnote }
684     \seq_put_right:Ne \l_@@_notes_labels_seq
685     {
686       \tl_if_novalue:nTF { #1 }
687       { \@@_notes_format:n { \int_use:N \c@tabularnote } }
688       { #1 }
689     }
690     \peek_meaning:NF \tabularnote
691     {
692       \@@_notes_label_in_tabular:n
693       { \seq_use:Nnnn \l_@@_notes_labels_seq { , } { , } { , } }
694       \seq_clear:N \l_@@_notes_labels_seq
695     }
696   }

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

```

7 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.


```

699 \cs_new_protected:Npn \@@_pgf_rect_node:nnnnn #1 #2 #3 #4 #5
700 {
701   \begin { pgfscope }
702   \pgfset
703   {
704     inner~sep = \c_zero_dim ,
705     minimum~size = \c_zero_dim
706   }
707   \pgftransformshift { \pgfpoint { 0.5 * ( #2 + #4 ) } { 0.5 * ( #3 + #5 ) } }
708   \pgfnode
709   { rectangle }
710   { center }
711   {
712     \vbox_to_ht:nn
713     { \dim_abs:n { #5 - #3 } }
714     {
715       \vfill
716       \hbox_to_wd:nn { \dim_abs:n { #4 - #2 } } { }
717     }
718   }
719   { #1 }
720   { }
721   \end { pgfscope }
722 }

```

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.

```

723 \cs_new_protected:Npn \@@_pgf_rect_node:nnn #1 #2 #3
724 {
725   \begin { pgfscope }
726   \pgfset
727   {
728     inner~sep = \c_zero_dim ,
729     minimum~size = \c_zero_dim
730   }
731   \pgftransformshift { \pgfpointscale { 0.5 } { \pgfpointadd { #2 } { #3 } } }
732   \pgfpointdiff { #3 } { #2 }
733   \pgfgetlastxy \l_tmpa_dim \l_tmpb_dim
734   \pgfnode
735   { rectangle }
736   { center }
737   {
738     \vbox_to_ht:nn
739     { \dim_abs:n \l_tmpb_dim }
740     { \vfill \hbox_to_wd:nn { \dim_abs:n \l_tmpa_dim } { } }
741   }
742   { #1 }
743   { }
744   \end { pgfscope }
745 }

```

8 The options

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

```

746 \tl_new:N \l_@@_caption_tl
747 \tl_new:N \l_@@_short_caption_tl
748 \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).

```
749 \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.

```
750 \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`.

```
751 \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`).

```
752 \dim_new:N \l_@@_cell_space_top_limit_dim
753 \dim_new:N \l_@@_cell_space_bottom_limit_dim
```

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

```
754 \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.

```
755 \dim_new:N \l_@@_xdots_inter_dim
756 \hook_gput_code:nnn { begindocument } { . }
757 { \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).

```
758 \dim_new:N \l_@@_xdots_shorten_start_dim
759 \dim_new:N \l_@@_xdots_shorten_end_dim
760 \hook_gput_code:nnn { begindocument } { . }
761 {
762   \dim_set:Nn \l_@@_xdots_shorten_start_dim { 0.3 em }
763   \dim_set:Nn \l_@@_xdots_shorten_end_dim { 0.3 em }
764 }
```

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.

```
765 \dim_new:N \l_@@_xdots_radius_dim
766 \hook_gput_code:nnn { begindocument } { . }
767 { \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.

```
768 \tl_new:N \l_@@_xdots_line_style_tl
769 \tl_const:Nn \c_@@_standard_tl { standard }
770 \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`.

```
771 \bool_new:N \l_@@_light_syntax_bool
772 \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).

```
773 \tl_new:N \l_@@_baseline_tl
774 \tl_set:Nn \l_@@_baseline_tl { c }
```

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

```
775 \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`).

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

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

```
777 \bool_new:N \l_@@_parallelize_diags_bool
778 \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.

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

```
780 \dim_new:N \l_@@_notes_above_space_dim
781 \hook_gput_code:nnn { begindocument } { . }
782 { \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.).

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

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

```
784 \cs_new_protected:Npn \@@_reset_arraystretch:
785 { \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).

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

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

```
787 \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.

```
788 \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”.

```
789 \bool_new:N \l_@@_medium_nodes_bool
790 \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).

```
791 \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`).

```
792 \dim_new:N \l_@@_left_margin_dim
793 \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`.

```
794 \dim_new:N \l_@@_extra_left_margin_dim
795 \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.

```
796 \tl_new:N \l_@@_end_of_row_tl
797 \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 “:”.

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

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

```
799 \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.

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

```
801 \keys_define:nn { nicematrix / xdots }
802 {
803   shorten-start .code:n =
804     \hook_gput_code:nnn { begindocument } { . }
805     { \dim_set:Nn \l_@@_xdots_shorten_start_dim { #1 } } ,
806   shorten-end .code:n =
807     \hook_gput_code:nnn { begindocument } { . }
808     { \dim_set:Nn \l_@@_xdots_shorten_end_dim { #1 } } ,
809   shorten-start .value_required:n = true ,
810   shorten-end .value_required:n = true ,
811   shorten .code:n =
812     \hook_gput_code:nnn { begindocument } { . }
813     {
814       \dim_set:Nn \l_@@_xdots_shorten_start_dim { #1 }
815       \dim_set:Nn \l_@@_xdots_shorten_end_dim { #1 }
816     } ,
817   shorten .value_required:n = true ,
```

```

818 horizontal-labels .bool_set:N = \l_@@_xdots_h_labels_bool ,
819 horizontal-labels .default:n = true ,
820 line-style .code:n =
821 {
822   \bool_lazy_or:nnTF
823   { \cs_if_exist_p:N \tikzpicture }
824   { \str_if_eq_p:nn { #1 } { standard } }
825   { \tl_set:Nn \l_@@_xdots_line_style_tl { #1 } }
826   { \@@_error:n { bad~option~for~line~style } }
827 } ,
828 line-style .value_required:n = true ,
829 color .tl_set:N = \l_@@_xdots_color_tl ,
830 color .value_required:n = true ,
831 radius .code:n =
832   \hook_gput_code:nnn { begindocument } { . }
833   { \dim_set:Nn \l_@@_xdots_radius_dim { #1 } } ,
834 radius .value_required:n = true ,
835 inter .code:n =
836   \hook_gput_code:nnn { begindocument } { . }
837   { \dim_set:Nn \l_@@_xdots_inter_dim { #1 } } ,
838 radius .value_required:n = true ,

```

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

```

839 down .code:n = \tl\_put\_right:Nn \l_@@_xdots_down_tl { #1 } ,
840 up .code:n = \tl\_put\_right:Nn \l_@@_xdots_up_tl { #1 } ,
841 middle .code:n = \tl\_put\_right:Nn \l_@@_xdots_middle_tl { #1 } ,

```

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

```

842 draw-first .code:n = \prg_do_nothing: ,
843 unknown .code:n = \@@_error:n { Unknown~key~for~xdots }
844 }

```

```

845 \keys_define:nn { nicematrix / rules }
846 {
847   color .tl_set:N = \l_@@_rules_color_tl ,
848   color .value_required:n = true ,
849   width .dim_set:N = \arrayrulewidth ,
850   width .value_required:n = true ,
851   unknown .code:n = \@@_error:n { Unknown~key~for~rules }
852 }

```

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

```

853 \keys_define:nn { nicematrix / Global }
854 {
855   ampersand-in-blocks .bool_set:N = \l_@@_amp_in_blocks_bool ,
856   ampersand-in-blocks .default:n = true ,
857   \&-in-blocks .meta:n = ampersand-in-blocks ,
858   no-cell-nodes .code:n =
859     \cs_set_protected:Npn \@@_node_for_cell:
860       { \box_use_drop:N \l_@@_cell_box } ,
861   no-cell-nodes .value_forbidden:n = true ,
862   rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
863   rounded-corners .default:n = 4 pt ,
864   custom-line .code:n = \@@_custom_line:n { #1 } ,
865   rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
866   rules .value_required:n = true ,
867   standard-cline .bool_set:N = \l_@@_standard_cline_bool ,

```

```

868 standard-cline .default:n = true ,
869 cell-space-top-limit .dim_set:N = \l_@@_cell_space_top_limit_dim ,
870 cell-space-top-limit .value_required:n = true ,
871 cell-space-bottom-limit .dim_set:N = \l_@@_cell_space_bottom_limit_dim ,
872 cell-space-bottom-limit .value_required:n = true ,
873 cell-space-limits .meta:n =
874 {
875     cell-space-top-limit = #1 ,
876     cell-space-bottom-limit = #1 ,
877 } ,
878 cell-space-limits .value_required:n = true ,
879 xdots .code:n = \keys_set:nn { nicematrix / xdots } { #1 } ,
880 light-syntax .code:n =
881     \bool_set_true:N \l_@@_light_syntax_bool
882     \bool_set_false:N \l_@@_light_syntax_expanded_bool ,
883 light-syntax .value_forbidden:n = true ,
884 light-syntax-expanded .code:n =
885     \bool_set_true:N \l_@@_light_syntax_bool
886     \bool_set_true:N \l_@@_light_syntax_expanded_bool ,
887 light-syntax-expanded .value_forbidden:n = true ,
888 end-of-row .tl_set:N = \l_@@_end_of_row_tl ,
889 end-of-row .value_required:n = true ,
890 first-col .code:n = \int_zero:N \l_@@_first_col_int ,
891 first-row .code:n = \int_zero:N \l_@@_first_row_int ,
892 last-row .int_set:N = \l_@@_last_row_int ,
893 last-row .default:n = -1 ,
894 code-for-first-col .tl_set:N = \l_@@_code_for_first_col_tl ,
895 code-for-first-col .value_required:n = true ,
896 code-for-last-col .tl_set:N = \l_@@_code_for_last_col_tl ,
897 code-for-last-col .value_required:n = true ,
898 code-for-first-row .tl_set:N = \l_@@_code_for_first_row_tl ,
899 code-for-first-row .value_required:n = true ,
900 code-for-last-row .tl_set:N = \l_@@_code_for_last_row_tl ,
901 code-for-last-row .value_required:n = true ,
902 hlines .clist_set:N = \l_@@_hlines_clist ,
903 vlines .clist_set:N = \l_@@_vlines_clist ,
904 hlines .default:n = all ,
905 vlines .default:n = all ,
906 vlines-in-sub-matrix .code:n =
907 {
908     \tl_if_single_token:nTF { #1 }
909     {
910         \tl_if_in:NnTF \c_@@_forbidden_letters_tl { #1 }
911         { \@@_error:nn { Forbidden-letter } { #1 } }

```

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

```

912     { \cs_set_eq:cN { @@_#1 } \@@_make_preamble_vlism:n }
913 }
914 { \@@_error:n { One-letter-allowed } }
915 } ,
916 vlines-in-sub-matrix .value_required:n = true ,
917 hvlines .code:n =
918 {
919     \bool_set_true:N \l_@@_hvlines_bool
920     \tl_set_eq:NN \l_@@_vlines_clist \c_@@_all_tl
921     \tl_set_eq:NN \l_@@_hlines_clist \c_@@_all_tl
922 } ,
923 hvlines-except-borders .code:n =
924 {
925     \tl_set_eq:NN \l_@@_vlines_clist \c_@@_all_tl
926     \tl_set_eq:NN \l_@@_hlines_clist \c_@@_all_tl
927     \bool_set_true:N \l_@@_hvlines_bool
928     \bool_set_true:N \l_@@_except_borders_bool
929 } ,

```

```

930 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.

```

931 renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
932 renew-dots .value_forbidden:n = true ,
933 nullify-dots .bool_set:N = \l_@@_nullify_dots_bool ,
934 create-medium-nodes .bool_set:N = \l_@@_medium_nodes_bool ,
935 create-large-nodes .bool_set:N = \l_@@_large_nodes_bool ,
936 create-extra-nodes .meta:n =
937   { create-medium-nodes , create-large-nodes } ,
938 left-margin .dim_set:N = \l_@@_left_margin_dim ,
939 left-margin .default:n = \arraycolsep ,
940 right-margin .dim_set:N = \l_@@_right_margin_dim ,
941 right-margin .default:n = \arraycolsep ,
942 margin .meta:n = { left-margin = #1 , right-margin = #1 } ,
943 margin .default:n = \arraycolsep ,
944 extra-left-margin .dim_set:N = \l_@@_extra_left_margin_dim ,
945 extra-right-margin .dim_set:N = \l_@@_extra_right_margin_dim ,
946 extra-margin .meta:n =
947   { extra-left-margin = #1 , extra-right-margin = #1 } ,
948 extra-margin .value_required:n = true ,
949 respect-arraystretch .code:n =
950   \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
951 respect-arraystretch .value_forbidden:n = true ,
952 pgf-node-code .tl_set:N = \l_@@_pgf_node_code_tl ,
953 pgf-node-code .value_required:n = true
954 }

```

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

```

955 \keys_define:nn { nicematrix / environments }
956 {
957   corners .clist_set:N = \l_@@_corners_clist ,
958   corners .default:n = { NW , SW , NE , SE } ,
959   code-before .code:n =
960     {
961       \tl_if_empty:nF { #1 }
962       {
963         \tl_gput_left:Nn \g_@@_pre_code_before_tl { #1 }
964         \bool_set_true:N \l_@@_code_before_bool
965       }
966     } ,
967   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}`.

```

968 c .code:n = \tl_set:Nn \l_@@_baseline_tl c ,
969 t .code:n = \tl_set:Nn \l_@@_baseline_tl t ,
970 b .code:n = \tl_set:Nn \l_@@_baseline_tl b ,
971 baseline .tl_set:N = \l_@@_baseline_tl ,
972 baseline .value_required:n = true ,
973 columns-width .code:n =
974   \tl_if_eq:nnTF { #1 } { auto }
975     { \bool_set_true:N \l_@@_auto_columns_width_bool }
976     { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,
977 columns-width .value_required:n = true ,
978 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.

```

979 \legacy_if:nF { measuring@ }

```

```

980     {
981         \str_set:Ne \l_tmpa_str { #1 }
982         \seq_if_in:NVTF \g_@@_names_seq \l_tmpa_str
983         { \@@_error:nn { Duplicate~name } { #1 } }
984         { \seq_gput_left:No \g_@@_names_seq \l_tmpa_str }
985         \str_set_eq:NN \l_@@_name_str \l_tmpa_str
986     } ,
987     name .value_required:n = true ,
988     code-after .tl_gset:N = \g_nicematrix_code_after_tl ,
989     code-after .value_required:n = true ,
990     color-inside .code:n =
991         \bool_set_true:N \l_@@_color_inside_bool
992         \bool_set_true:N \l_@@_code_before_bool ,
993     color-inside .value_forbidden:n = true ,
994     colortbl-like .meta:n = color-inside
995 }
996 \keys_define:nn { nicematrix / notes }
997 {
998     para .bool_set:N = \l_@@_notes_para_bool ,
999     para .default:n = true ,
1000     code-before .tl_set:N = \l_@@_notes_code_before_tl ,
1001     code-before .value_required:n = true ,
1002     code-after .tl_set:N = \l_@@_notes_code_after_tl ,
1003     code-after .value_required:n = true ,
1004     bottomrule .bool_set:N = \l_@@_notes_bottomrule_bool ,
1005     bottomrule .default:n = true ,
1006     style .cs_set:Np = \@@_notes_style:n #1 ,
1007     style .value_required:n = true ,
1008     label-in-tabular .cs_set:Np = \@@_notes_label_in_tabular:n #1 ,
1009     label-in-tabular .value_required:n = true ,
1010     label-in-list .cs_set:Np = \@@_notes_label_in_list:n #1 ,
1011     label-in-list .value_required:n = true ,
1012     enumitem-keys .code:n =
1013     {
1014         \hook_gput_code:nnn { begindocument } { . }
1015         {
1016             \IfPackageLoadedT { enumitem }
1017             { \setlist* [ tabularnotes ] { #1 } }
1018         }
1019     } ,
1020     enumitem-keys .value_required:n = true ,
1021     enumitem-keys-para .code:n =
1022     {
1023         \hook_gput_code:nnn { begindocument } { . }
1024         {
1025             \IfPackageLoadedT { enumitem }
1026             { \setlist* [ tabularnotes* ] { #1 } }
1027         }
1028     } ,
1029     enumitem-keys-para .value_required:n = true ,
1030     detect-duplicates .bool_set:N = \l_@@_notes_detect_duplicates_bool ,
1031     detect-duplicates .default:n = true ,
1032     unknown .code:n = \@@_error:n { Unknown~key~for~notes }
1033 }
1034 \keys_define:nn { nicematrix / delimiters }
1035 {
1036     max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1037     max-width .default:n = true ,
1038     color .tl_set:N = \l_@@_delimiters_color_tl ,
1039     color .value_required:n = true ,
1040 }

```

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

environments).

```

1041 \keys_define:nn { nicematrix }
1042 {
1043   NiceMatrixOptions .inherit:n =
1044     { nicematrix / Global } ,
1045   NiceMatrixOptions / xdots .inherit:n = nicematrix / xdots ,
1046   NiceMatrixOptions / rules .inherit:n = nicematrix / rules ,
1047   NiceMatrixOptions / notes .inherit:n = nicematrix / notes ,
1048   NiceMatrixOptions / sub-matrix .inherit:n = nicematrix / sub-matrix ,
1049   SubMatrix / rules .inherit:n = nicematrix / rules ,
1050   CodeAfter / xdots .inherit:n = nicematrix / xdots ,
1051   CodeBefore / sub-matrix .inherit:n = nicematrix / sub-matrix ,
1052   CodeAfter / sub-matrix .inherit:n = nicematrix / sub-matrix ,
1053   NiceMatrix .inherit:n =
1054     {
1055       nicematrix / Global ,
1056       nicematrix / environments ,
1057     } ,
1058   NiceMatrix / xdots .inherit:n = nicematrix / xdots ,
1059   NiceMatrix / rules .inherit:n = nicematrix / rules ,
1060   NiceTabular .inherit:n =
1061     {
1062       nicematrix / Global ,
1063       nicematrix / environments
1064     } ,
1065   NiceTabular / xdots .inherit:n = nicematrix / xdots ,
1066   NiceTabular / rules .inherit:n = nicematrix / rules ,
1067   NiceTabular / notes .inherit:n = nicematrix / notes ,
1068   NiceArray .inherit:n =
1069     {
1070       nicematrix / Global ,
1071       nicematrix / environments ,
1072     } ,
1073   NiceArray / xdots .inherit:n = nicematrix / xdots ,
1074   NiceArray / rules .inherit:n = nicematrix / rules ,
1075   pNiceArray .inherit:n =
1076     {
1077       nicematrix / Global ,
1078       nicematrix / environments ,
1079     } ,
1080   pNiceArray / xdots .inherit:n = nicematrix / xdots ,
1081   pNiceArray / rules .inherit:n = nicematrix / rules ,
1082 }

```

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

```

1083 \keys_define:nn { nicematrix / NiceMatrixOptions }
1084 {
1085   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1086   delimiters / color .value_required:n = true ,
1087   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1088   delimiters / max-width .default:n = true ,
1089   delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
1090   delimiters .value_required:n = true ,
1091   width .dim_set:N = \l_@@_width_dim ,
1092   width .value_required:n = true ,
1093   last-col .code:n =
1094     \tl_if_empty:nF { #1 }
1095     { \@@_error:n { last-col~non-empty~for~NiceMatrixOptions } }
1096     \int_zero:N \l_@@_last_col_int ,
1097   small .bool_set:N = \l_@@_small_bool ,
1098   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.

```
1099   renew-matrix .code:n = \@@_renew_matrix: ,
1100   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}`.

```
1101   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.

```
1102   columns-width .code:n =
1103     \tl_if_eq:nnTF { #1 } { auto }
1104     { \@@_error:n { Option~auto~for~columns~width } }
1105     { \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.

```
1106   allow-duplicate-names .code:n =
1107     \@@_msg_redirect_name:nn { Duplicate~name } { none } ,
1108   allow-duplicate-names .value_forbidden:n = true ,
1109   notes .code:n = \keys_set:nn { nicematrix / notes } { #1 } ,
1110   notes .value_required:n = true ,
1111   sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1112   sub-matrix .value_required:n = true ,
1113   matrix / columns-type .tl_set:N = \l_@@_columns_type_tl ,
1114   matrix / columns-type .value_required:n = true ,
1115   caption-above .bool_set:N = \l_@@_caption_above_bool ,
1116   caption-above .default:n = true ,
1117   unknown .code:n = \@@_error:n { Unknown~key~for~NiceMatrixOptions }
1118 } ,
```

`\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.

```
1119 \NewDocumentCommand \NiceMatrixOptions { m }
1120 { \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.

```
1121 \keys_define:nn { nicematrix / NiceMatrix }
1122 {
1123   last-col .code:n = \tl_if_empty:nTF { #1 }
1124     {
1125       \bool_set_true:N \l_@@_last_col_without_value_bool
1126       \int_set:Nn \l_@@_last_col_int { -1 }
1127     }
1128     { \int_set:Nn \l_@@_last_col_int { #1 } } ,
1129   columns-type .tl_set:N = \l_@@_columns_type_tl ,
1130   columns-type .value_required:n = true ,
1131   l .meta:n = { columns-type = l } ,
1132   r .meta:n = { columns-type = r } ,
1133   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1134   delimiters / color .value_required:n = true ,
1135   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1136   delimiters / max-width .default:n = true ,
1137   delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
1138   delimiters .value_required:n = true ,
1139   small .bool_set:N = \l_@@_small_bool ,
```

```

1140     small .value_forbidden:n = true ,
1141     unknown .code:n = \@@_error:n { Unknown~key~for~NiceMatrix }
1142 }

```

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

```

1143 \keys_define:nn { nicematrix / NiceArray }
1144 {

```

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.

```

1145     small .bool_set:N = \l_@@_small_bool ,
1146     small .value_forbidden:n = true ,
1147     last-col .code:n = \tl_if_empty:nF { #1 }
1148         { \@@_error:n { last-col-non~empty~for~NiceArray } }
1149         \int_zero:N \l_@@_last_col_int ,
1150     r .code:n = \@@_error:n { r-or~l~with~preamble } ,
1151     l .code:n = \@@_error:n { r-or~l~with~preamble } ,
1152     unknown .code:n = \@@_error:n { Unknown~key~for~NiceArray }
1153 }

1154 \keys_define:nn { nicematrix / pNiceArray }
1155 {
1156     first-col .code:n = \int_zero:N \l_@@_first_col_int ,
1157     last-col .code:n = \tl_if_empty:nF { #1 }
1158         { \@@_error:n { last-col-non~empty~for~NiceArray } }
1159         \int_zero:N \l_@@_last_col_int ,
1160     first-row .code:n = \int_zero:N \l_@@_first_row_int ,
1161     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1162     delimiters / color .value_required:n = true ,
1163     delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1164     delimiters / max-width .default:n = true ,
1165     delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
1166     delimiters .value_required:n = true ,
1167     small .bool_set:N = \l_@@_small_bool ,
1168     small .value_forbidden:n = true ,
1169     r .code:n = \@@_error:n { r-or~l~with~preamble } ,
1170     l .code:n = \@@_error:n { r-or~l~with~preamble } ,
1171     unknown .code:n = \@@_error:n { Unknown~key~for~NiceMatrix }
1172 }

```

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

```

1173 \keys_define:nn { nicematrix / NiceTabular }
1174 {

```

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.

```

1175     width .code:n = \dim_set:Nn \l_@@_width_dim { #1 }
1176         \bool_set_true:N \l_@@_width_used_bool ,
1177     width .value_required:n = true ,
1178     notes .code:n = \keys_set:nn { nicematrix / notes } { #1 } ,
1179     tabularnote .tl_gset:N = \g_@@_tabularnote_tl ,
1180     tabularnote .value_required:n = true ,
1181     caption .tl_set:N = \l_@@_caption_tl ,
1182     caption .value_required:n = true ,
1183     short-caption .tl_set:N = \l_@@_short_caption_tl ,
1184     short-caption .value_required:n = true ,
1185     label .tl_set:N = \l_@@_label_tl ,
1186     label .value_required:n = true ,
1187     last-col .code:n = \tl_if_empty:nF { #1 }
1188         { \@@_error:n { last-col-non~empty~for~NiceArray } }

```

```

1189             \int_zero:N \l_@@_last_col_int ,
1190     r .code:n = \@@_error:n { r-or~l-with-preamble } ,
1191     l .code:n = \@@_error:n { r-or~l-with-preamble } ,
1192     unknown .code:n = \@@_error:n { Unknown~key~for~NiceTabular }
1193 }

```

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
```

```

1194 \keys_define:nn { nicematrix / CodeAfter }
1195 {
1196     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1197     delimiters / color .value_required:n = true ,
1198     rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
1199     rules .value_required:n = true ,
1200     xdots .code:n = \keys_set:nn { nicematrix / xdots } { #1 } ,
1201     sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1202     sub-matrix .value_required:n = true ,
1203     unknown .code:n = \@@_error:n { Unknown~key~for~CodeAfter }
1204 }

```

9 Important code used by {NiceArrayWithDelims}

The pseudo-environment `\@@_cell_begin:w-\@@_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}`).

```

1205 \cs_new_protected:Npn \@@_cell_begin:w
1206 {

```

`\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.

```
1207     \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).

```
1208     \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
```

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

```
1209     \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.

```

1210     \int_compare:nNnT \c@jCol = \c_one_int
1211     { \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:`.

```
1212     \hbox_set:Nw \l_@@_cell_box
```

The following command is nullified in the tabulars.

```

1213     \@@_tuning_not_tabular_begin:
1214     \@@_tuning_first_row:
1215     \@@_tuning_last_row:
1216     \g_@@_row_style_tl
1217 }

```

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

```

1218 \cs_new_protected:Npn \@@_tuning_first_row:
1219 {
1220   \int_if_zero:nT \c@iRow
1221   {
1222     \int_compare:nNnT \c@jCol > \c_zero_int
1223     {
1224       \l_@@_code_for_first_row_tl
1225       \xglobal \colorlet { nicematrix-first-row } { . }
1226     }
1227   }
1228 }

```

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

```

1229 \cs_new_protected:Npn \@@_tuning_last_row:
1230 {
1231   \int_compare:nNnT \c@iRow = \l_@@_last_row_int
1232   {
1233     \l_@@_code_for_last_row_tl
1234     \xglobal \colorlet { nicematrix-last-row } { . }
1235   }
1236 }

```

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

```

1237 \cs_set_eq:NN \@@_tuning_key_small: \prg_do_nothing:

```

The following commands are nullified in the tabulars.

```

1238 \cs_set_nopar:Npn \@@_tuning_not_tabular_begin:
1239 {
1240   \c_math_toggle_token

```

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

```

1241   \@@_tuning_key_small:
1242 }
1243 \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.

```

1244 \cs_new_protected:Npn \@@_begin_of_row:
1245 {
1246   \int_gincr:N \c@iRow
1247   \dim_gset_eq:NN \g_@@_dp_ante_last_row_dim \g_@@_dp_last_row_dim
1248   \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
1249   \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1250   \pgfpicture
1251   \pgfrememberpicturerepositiononpagetrue
1252   \pgfcoordinate
1253   { \@@_env: - row - \int_use:N \c@iRow - base }
1254   { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
1255   \str_if_empty:NF \l_@@_name_str
1256   {
1257     \pgfnodealias
1258     { \l_@@_name_str - row - \int_use:N \c@iRow - base }
1259     { \@@_env: - row - \int_use:N \c@iRow - base }
1260   }
1261   \endpgfpicture
1262 }

```

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.

```

1263 \cs_new_protected:Npn \@@_update_for_first_and_last_row:
1264 {
1265   \int_if_zero:nTF \c@iRow
1266   {
1267     \dim_gset:Nn \g_@@_dp_row_zero_dim
1268     { \dim_max:nn \g_@@_dp_row_zero_dim { \box_dp:N \l_@@_cell_box } }
1269     \dim_gset:Nn \g_@@_ht_row_zero_dim
1270     { \dim_max:nn \g_@@_ht_row_zero_dim { \box_ht:N \l_@@_cell_box } }
1271   }
1272   {
1273     \int_compare:nNnT \c@iRow = \c_one_int
1274     {
1275       \dim_gset:Nn \g_@@_ht_row_one_dim
1276       { \dim_max:nn \g_@@_ht_row_one_dim { \box_ht:N \l_@@_cell_box } }
1277     }
1278   }
1279 }

1280 \cs_new_protected:Npn \@@_rotate_cell_box:
1281 {
1282   \box_rotate:Nn \l_@@_cell_box { 90 }
1283   \bool_if:NTF \g_@@_rotate_c_bool
1284   {
1285     \hbox_set:Nn \l_@@_cell_box
1286     {
1287       \c_math_toggle_token
1288       \vcenter { \box_use:N \l_@@_cell_box }
1289       \c_math_toggle_token
1290     }
1291   }
1292   {
1293     \int_compare:nNnT \c@iRow = \l_@@_last_row_int
1294     {
1295       \vbox_set_top:Nn \l_@@_cell_box
1296       {
1297         \vbox_to_zero:n { }
1298         \skip_vertical:n { - \box_ht:N \@arstrutbox + 0.8 ex }
1299         \box_use:N \l_@@_cell_box
1300       }
1301     }
1302   }
1303   \bool_gset_false:N \g_@@_rotate_bool
1304   \bool_gset_false:N \g_@@_rotate_c_bool
1305 }

1306 \cs_new_protected:Npn \@@_adjust_size_box:
1307 {
1308   \dim_compare:nNnT \g_@@_blocks_wd_dim > \c_zero_dim
1309   {
1310     \box_set_wd:Nn \l_@@_cell_box
1311     { \dim_max:nn { \box_wd:N \l_@@_cell_box } \g_@@_blocks_wd_dim }
1312     \dim_gzero:N \g_@@_blocks_wd_dim
1313   }
1314   \dim_compare:nNnT \g_@@_blocks_dp_dim > \c_zero_dim
1315   {
1316     \box_set_dp:Nn \l_@@_cell_box
1317     { \dim_max:nn { \box_dp:N \l_@@_cell_box } \g_@@_blocks_dp_dim }
1318     \dim_gzero:N \g_@@_blocks_dp_dim
1319   }
1320   \dim_compare:nNnT \g_@@_blocks_ht_dim > \c_zero_dim
1321   {

```

```

1322     \box_set_ht:Nn \l_@@_cell_box
1323     { \dim_max:nn { \box_ht:N \l_@@_cell_box } \g_@@_blocks_ht_dim }
1324     \dim_gzero:N \g_@@_blocks_ht_dim
1325   }
1326 }
1327 \cs_new_protected:Npn \@@_cell_end:
1328 {

```

The following command is nullified in the tabulars.

```

1329     \@@_tuning_not_tabular_end:
1330     \hbox_set_end:
1331     \@@_cell_end_i:
1332   }
1333 \cs_new_protected:Npn \@@_cell_end_i:
1334 {

```

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.

```

1335     \g_@@_cell_after_hook_tl
1336     \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
1337     \@@_adjust_size_box:
1338     \box_set_ht:Nn \l_@@_cell_box
1339     { \box_ht:N \l_@@_cell_box + \l_@@_cell_space_top_limit_dim }
1340     \box_set_dp:Nn \l_@@_cell_box
1341     { \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”).

```

1342     \@@_update_max_cell_width:

```

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

```

1343     \@@_update_for_first_and_last_row:

```

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

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

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

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

```

1344     \bool_if:NTF \g_@@_empty_cell_bool
1345     { \box_use_drop:N \l_@@_cell_box }
1346     {
1347         \bool_if:NTF \g_@@_not_empty_cell_bool
1348         \@@_node_for_cell:
1349         {
1350             \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > \c_zero_dim
1351             \@@_node_for_cell:

```

```

1352         { \box_use_drop:N \l_@@_cell_box }
1353     }
1354 }
1355 \int_gset:Nn \g_@@_col_total_int { \int_max:nn \g_@@_col_total_int \c@jCol }
1356 \bool_gset_false:N \g_@@_empty_cell_bool
1357 \bool_gset_false:N \g_@@_not_empty_cell_bool
1358 }

```

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

```

1359 \cs_new_protected:Npn \@@_update_max_cell_width:
1360 {
1361     \dim_gset:Nn \g_@@_max_cell_width_dim
1362     { \dim_max:nn \g_@@_max_cell_width_dim { \box_wd:N \l_@@_cell_box } }
1363 }

```

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`).

```

1364 \cs_new_protected:Npn \@@_cell_end_for_w_s:
1365 {
1366     \@@_math_toggle:
1367     \hbox_set_end:
1368     \bool_if:NF \g_@@_rotate_bool
1369     {
1370         \hbox_set:Nn \l_@@_cell_box
1371         {
1372             \makebox [ \l_@@_col_width_dim ] [ s ]
1373             { \hbox_unpack_drop:N \l_@@_cell_box }
1374         }
1375     }
1376     \@@_cell_end_i:
1377 }

```

```

1378 \pgfset
1379 {
1380     nicematrix / cell-node /.style =
1381     {
1382         inner-sep = \c_zero_dim ,
1383         minimum-width = \c_zero_dim
1384     }
1385 }

```

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

```

1386 \cs_new_protected:Npn \@@_node_for_cell:
1387 {
1388     \pgfpicture
1389     \pgfsetbaseline \c_zero_dim
1390     \pgfrememberpicturepositiononpagetrue
1391     \pgfset { nicematrix / cell-node }
1392     \pgfnode
1393     { rectangle }
1394     { base }
1395     {

```

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

```

1396     \set@color
1397     \box_use_drop:N \l_@@_cell_box
1398 }
1399 { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1400 { \l_@@_pgf_node_code_tl }

```



```

1401 \str_if_empty:NF \l_@@_name_str
1402 {
1403   \pgfnodealias
1404     { \l_@@_name_str - \int_use:N \c@iRow - \int_use:N \c@jCol }
1405     { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1406 }
1407 \endpgfpicture
1408 }

```

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.

```

1409 \cs_new_protected:Npn \@@_patch_node_for_cell:n #1
1410 {
1411   \cs_new_protected:Npn \@@_patch_node_for_cell:
1412   {
1413     \hbox_set:Nn \l_@@_cell_box
1414     {
1415       \box_move_up:nn { \box_ht:N \l_@@_cell_box }
1416       \hbox_overlap_left:n
1417       {
1418         \pgfsys@markposition
1419         { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - NW }

```

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

```

1420       #1
1421     }
1422     \box_use:N \l_@@_cell_box
1423     \box_move_down:nn { \box_dp:N \l_@@_cell_box }
1424     \hbox_overlap_left:n
1425     {
1426       \pgfsys@markposition
1427       { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - SE }
1428     }
1429   }
1430 }
1431 }
1432 }

```

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

```

1433 \bool_lazy_or:nnTF \sys_if_engine_xetex_p: \sys_if_output_dvi_p:
1434 {
1435   \@@_patch_node_for_cell:n
1436   { \skip_horizontal:n { 0.5 \box_wd:N \l_@@_cell_box } }
1437 }
1438 { \@@_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).

```
1439 \cs_new_protected:Npn \@@_instruction_of_type:nnn #1 #2 #3
1440 {
1441   \bool_if:nTF { #1 } \tl_gput_left:ce \tl_gput_right:ce
1442   { g_@@_ #2 _ lines _ tl }
1443   {
1444     \use:c { @@ _ draw _ #2 : nnn }
1445     { \int_use:N \c@iRow }
1446     { \int_use:N \c@jCol }
1447     { \exp_not:n { #3 } }
1448   }
1449 }

1450 \cs_generate_variant:Nn \@@_array:n { o }
1451 \cs_new_protected:Npn \@@_array:n
1452 {
1453   % \begin{macrocode}
1454   \dim_set:Nn \col@sep
1455   { \bool_if:nTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
1456   \dim_compare:nNnTF \l_@@_tabular_width_dim = \c_zero_dim
1457   { \cs_set_nopar:Npn \@halignto { } }
1458   { \cs_set_nopar:Npx \@halignto { to \dim_use:N \l_@@_tabular_width_dim } }
```

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

```
1459   \@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:onTF is fully expandable and we need something fully expandable here.

1460   [ \str_if_eq:onTF \l_@@_baseline_tl c c t ]
1461 }
```

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.

```
1462 \bool_if:nTF \c_@@_tagging_array_bool
1463 { \cs_set_eq:NN \@@_old_ar@ialign: \ar@ialign }
1464 { \cs_set_eq:NN \@@_old_ialign: \ialign }
```

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

```
1465 \cs_new_protected:Npn \@@_create_row_node:
1466 {
1467   \int_compare:nNnT \c@iRow > \g_@@_last_row_node_int
1468   {
1469     \int_gset_eq:NN \g_@@_last_row_node_int \c@iRow
1470     \@@_create_row_node_i:
1471   }
1472 }

1473 \cs_new_protected:Npn \@@_create_row_node_i:
1474 {
```

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

```
1475   \hbox
1476   {
1477     \bool_if:NT \l_@@_code_before_bool
1478     {
1479       \vtop
```

```

1480         {
1481             \skip_vertical:N 0.5\arrayrulewidth
1482             \pgfsys@markposition
1483             { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1484             \skip_vertical:N -0.5\arrayrulewidth
1485         }
1486     }
1487     \pgfpicture
1488     \pgfrememberpicturepositiononpagetrue
1489     \pgfcoordinate { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1490     { \pgfpoint \c_zero_dim { - 0.5 \arrayrulewidth } }
1491     \str_if_empty:NF \l_@@_name_str
1492     {
1493         \pgfnodealias
1494         { \l_@@_name_str - row - \int_eval:n { \c@iRow + 1 } }
1495         { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1496     }
1497     \endpgfpicture
1498 }
1499 }

```

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

```

1500 \cs_new:Npn \@@_everycr: { \noalign { \@@_everycr_i: } }

1501 \cs_new_protected:Npn \@@_everycr_i:
1502 {
1503     \bool_if:NT \c_@@_testphase_table_bool
1504     {
1505         \tbl_if_row_was_started:T { \UseTaggingSocket { tbl / row / end } }
1506         \tbl_update_cell_data_for_next_row:
1507     }
1508     \int_gzero:N \c@jCol
1509     \bool_gset_false:N \g_@@_after_col_zero_bool
1510     \bool_if:NF \g_@@_row_of_col_done_bool
1511     {
1512         \@@_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).

```

1513     \tl_if_empty:NF \l_@@_hlines_clist
1514     {
1515         \tl_if_eq:NNF \l_@@_hlines_clist \c_@@_all_tl
1516         {
1517             \clist_if_in:NeT
1518             \l_@@_hlines_clist
1519             { \int_eval:n { \c@iRow + 1 } }
1520         }
1521     }

```

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.

```

1522     \int_compare:nNnT \c@iRow > { -1 }
1523     {
1524         \int_compare:nNnF \c@iRow = \l_@@_last_row_int
1525         { \hrule height \arrayrulewidth width \c_zero_dim }
1526     }
1527 }
1528 }
1529 }
1530 }

```

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

```

1531 \cs_set_protected:Npn \@@_renew_dots:
1532 {
1533   \cs_set_eq:NN \ldots \@@_Ldots
1534   \cs_set_eq:NN \cdots \@@_Cdots
1535   \cs_set_eq:NN \vdots \@@_Vdots
1536   \cs_set_eq:NN \ddots \@@_Ddots
1537   \cs_set_eq:NN \iddots \@@_Iddots
1538   \cs_set_eq:NN \dots \@@_Ldots
1539   \cs_set_eq:NN \hdotsfor \@@_Hdotsfor:
1540 }
1541 \cs_new_protected:Npn \@@_test_color_inside:
1542 {
1543   \bool_if:NF \l_@@_color_inside_bool
1544   {

```

We will issue an error only during the first run.

```

1545     \bool_if:NF \g_@@_aux_found_bool
1546     { \@@_error:n { without~color-inside } }
1547   }
1548 }
1549 \cs_new_protected:Npn \@@_redefine_everycr:
1550 { \everycr { \@@_everycr: } }
1551 \hook_gput_code:nnn { begindocument } { . }
1552 {
1553   \IfPackageLoadedT { colortbl }
1554   {
1555     \cs_set_protected:Npn \@@_redefine_everycr:
1556     {
1557       \CT@everycr
1558       {
1559         \noalign { \cs_gset_eq:NN \CT@row@color \prg_do_nothing: }
1560         \@@_everycr:
1561       }
1562     }
1563   }
1564 }

```

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.

```

1565 \hook_gput_code:nnn { begindocument } { . }
1566 {
1567   \IfPackageLoadedTF { booktabs }
1568   {
1569     \cs_new_protected:Npn \@@_patch_booktabs:
1570     { \tl_put_left:Nn \@BTnormal \@@_create_row_node_i: }
1571   }
1572   { \cs_new_protected:Npn \@@_patch_booktabs: { } }
1573 }

```

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

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

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

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

```

1574 \cs_new_protected:Npn \@_some_initialization:
1575 {
1576   \dim_gset:Nn \g_@@_dp_row_zero_dim { \box_dp:N \@arstrutbox }
1577   \dim_gset:Nn \g_@@_ht_row_zero_dim { \box_ht:N \@arstrutbox }
1578   \dim_gset_eq:NN \g_@@_ht_row_one_dim \g_@@_ht_row_zero_dim
1579   \dim_gzero:N \g_@@_dp_ante_last_row_dim
1580   \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1581   \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
1582 }

```

The following code `\@@_pre_array_ii:` is used in `{NiceArrayWithDelims}`. It exists as a standalone macro only for legibility.

```

1583 \cs_new_protected:Npn \@_pre_array_ii:
1584 {

```

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

```

1585   \int_gzero:N \g_@@_total_X_weight_int
1586   \@@_expand_clist:N \l_@@_hlines_clist
1587   \@@_expand_clist:N \l_@@_vlines_clist
1588   \@@_patch_booktabs:
1589   \box_clear_new:N \l_@@_cell_box
1590   \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}`).

```

1591   \bool_if:NT \l_@@_small_bool
1592   {
1593     \cs_set_nopar:Npn \arraystretch { 0.47 }
1594     \dim_set:Nn \arraycolsep { 1.45 pt }

```

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

```

1595     \cs_set_eq:NN \@_tuning_key_small: \scriptstyle
1596   }

```

```

1597   \bool_if:NT \g_@@_recreate_cell_nodes_bool
1598   {
1599     \tl_put_right:Nn \@_begin_of_row:
1600     {
1601       \pgfsys@markposition
1602       { \@_env: - row - \int_use:N \c@iRow - base }
1603     }
1604   }

```

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

```

1605   \bool_if:NTF \c_@@_tagging_array_bool
1606   {
1607     \cs_set_nopar:Npn \ar@ialign
1608     {
1609       \bool_if:NT \c_@@_testphase_table_bool \tbl_init_cell_data_for_table:
1610       \@@_redefine_everycr:
1611       \dim_zero:N \tabskip
1612       \@_some_initialization:

```

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

```

1613         \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign:
1614         \halign
1615     }
1616 }

```

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

```

1617 {
1618     \cs_set_nopar:Npn \ialign
1619     {
1620         \@@_redefine_everycr:
1621         \dim_zero:N \tabskip
1622         \@@_some_initialization:
1623         \cs_set_eq:NN \ialign \@@_old_ialign:
1624         \halign
1625     }
1626 }

```

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).

```

1627     \cs_set_eq:NN \@@_old_ldots \ldots
1628     \cs_set_eq:NN \@@_old_cdots \cdots
1629     \cs_set_eq:NN \@@_old_vdots \vdots
1630     \cs_set_eq:NN \@@_old_ddots \ddots
1631     \cs_set_eq:NN \@@_old_iddots \iddots
1632     \bool_if:NTF \l_@@_standard_cline_bool
1633     { \cs_set_eq:NN \cline \@@_standard_cline }
1634     { \cs_set_eq:NN \cline \@@_cline }
1635     \cs_set_eq:NN \Ldots \@@_Ldots
1636     \cs_set_eq:NN \Cdots \@@_Cdots
1637     \cs_set_eq:NN \Vdots \@@_Vdots
1638     \cs_set_eq:NN \Ddots \@@_Ddots
1639     \cs_set_eq:NN \Iddots \@@_Iddots
1640     \cs_set_eq:NN \Hline \@@_Hline:
1641     \cs_set_eq:NN \Hspace \@@_Hspace:
1642     \cs_set_eq:NN \Hdotsfor \@@_Hdotsfor:
1643     \cs_set_eq:NN \Vdotsfor \@@_Vdotsfor:
1644     \cs_set_eq:NN \Block \@@_Block:
1645     \cs_set_eq:NN \rotate \@@_rotate:
1646     \cs_set_eq:NN \OnlyMainNiceMatrix \@@_OnlyMainNiceMatrix:n
1647     \cs_set_eq:NN \dotfill \@@_dotfill:
1648     \cs_set_eq:NN \CodeAfter \@@_CodeAfter:
1649     \cs_set_eq:NN \diagbox \@@_diagbox:nn
1650     \cs_set_eq:NN \NotEmpty \@@_NotEmpty:
1651     \cs_set_eq:NN \RowStyle \@@_RowStyle:n
1652     \seq_map_inline:Nn \l_@@_custom_line_commands_seq
1653     { \cs_set_eq:cc { ##1 } { nicematrix - ##1 } }
1654     \cs_set_eq:NN \cellcolor \@@_cellcolor_tabular
1655     \cs_set_eq:NN \rowcolor \@@_rowcolor_tabular
1656     \cs_set_eq:NN \rowcolors \@@_rowcolors_tabular
1657     \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors_tabular
1658     \int_compare:nNnT \l_@@_first_row_int > \c_zero_int
1659     { \cs_set_eq:NN \@@_tuning_first_row: \prg_do_nothing: }
1660     \int_compare:nNnT \l_@@_last_row_int < \c_zero_int
1661     { \cs_set_eq:NN \@@_tuning_last_row: \prg_do_nothing: }
1662     \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:`.

```

1663     \cs_set_eq:NN \multicolumn \@@_multicolumn:nnn

```

```

1664 \hook_gput_code:nnn { env / tabular / begin } { nicematrix }
1665 { \cs_set_eq:NN \multicolumn \@_old_multicolumn }
1666 \@_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!).

```

1667 \tl_if_exist:NT \l_@@_note_in_caption_tl
1668 {
1669   \tl_if_empty:NF \l_@@_note_in_caption_tl
1670   {
1671     \int_gset_eq:NN \g_@@_notes_caption_int \l_@@_note_in_caption_tl
1672     \int_gset:Nn \c@tabularnote { \l_@@_note_in_caption_tl }
1673   }
1674 }

```

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

```

1675 \seq_gclear:N \g_@@_multicolumn_cells_seq
1676 \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).

```

1677 \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 of rows excepted the last row (if `\l_@@_last_row_bool` has been raised with the option `last-row`).

```

1678 \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:w` executed at the beginning of each cell.

```

1679 \int_gzero_new:N \g_@@_col_total_int
1680 \cs_set_eq:NN \@ifnextchar \new@ifnextchar
1681 \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.

```

1682 \tl_gclear_new:N \g_@@_Cdots_lines_tl
1683 \tl_gclear_new:N \g_@@_Ldots_lines_tl
1684 \tl_gclear_new:N \g_@@_Vdots_lines_tl
1685 \tl_gclear_new:N \g_@@_Ddots_lines_tl
1686 \tl_gclear_new:N \g_@@_Iddots_lines_tl
1687 \tl_gclear_new:N \g_@@_HVDotsfor_lines_tl

1688 \tl_gclear:N \g_nicematrix_code_before_tl
1689 \tl_gclear:N \g_@@_pre_code_before_tl
1690 }

```

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

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

```

1691 \cs_new_protected:Npn \@@_pre_array:
1692 {
1693   \cs_if_exist:NT \theiRow { \int_set_eq:NN \l_@@_old_iRow_int \c@iRow }
1694   \int_gzero_new:N \c@iRow
1695   \cs_if_exist:NT \thejCol { \int_set_eq:NN \l_@@_old_jCol_int \c@jCol }
1696   \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).

```

1697   \int_compare:nNnT \l_@@_last_row_int = { -1 }
1698   {
1699     \bool_set_true:N \l_@@_last_row_without_value_bool
1700     \bool_if:NT \g_@@_aux_found_bool
1701       { \int_set:Nn \l_@@_last_row_int { \seq_item:Nn \g_@@_size_seq 3 } }
1702   }
1703   \int_compare:nNnT \l_@@_last_col_int = { -1 }
1704   {
1705     \bool_if:NT \g_@@_aux_found_bool
1706       { \int_set:Nn \l_@@_last_col_int { \seq_item:Nn \g_@@_size_seq 6 } }
1707   }

```

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

```

1708   \int_compare:nNnT \l_@@_last_row_int > { -2 }
1709   {
1710     \tl_put_right:Nn \@@_update_for_first_and_last_row:
1711       {
1712         \dim_gset:Nn \g_@@_ht_last_row_dim
1713           { \dim_max:nn \g_@@_ht_last_row_dim { \box_ht:N \l_@@_cell_box } }
1714         \dim_gset:Nn \g_@@_dp_last_row_dim
1715           { \dim_max:nn \g_@@_dp_last_row_dim { \box_dp:N \l_@@_cell_box } }
1716       }
1717   }

1718   \seq_gclear:N \g_@@_cols_vlism_seq
1719   \seq_gclear:N \g_@@_submatrix_seq

```

Now the `\CodeBefore`.

```

1720   \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.

```

1721   \seq_gclear:N \g_@@_pos_of_blocks_seq

```

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

```

1722   \seq_gclear_new:N \g_@@_multicolumn_cells_seq
1723   \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.

```

1724   \int_gset:Nn \g_@@_last_row_node_int { -2 }

```

The value `-2` is important.

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

```

1725   \@@_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.

```
1726 \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]`).

```
1727 \dim_zero_new:N \l_@@_left_delim_dim
1728 \dim_zero_new:N \l_@@_right_delim_dim
1729 \bool_if:NTF \g_@@_delims_bool
1730 {
```

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

```
1731 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_left_delim_tl $ }
1732 \dim_set:Nn \l_@@_left_delim_dim { \box_wd:N \l_tmpa_box }
1733 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_right_delim_tl $ }
1734 \dim_set:Nn \l_@@_right_delim_dim { \box_wd:N \l_tmpa_box }
1735 }
1736 {
1737 \dim_gset:Nn \l_@@_left_delim_dim
1738 { 2 \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
1739 \dim_gset_eq:NN \l_@@_right_delim_dim \l_@@_left_delim_dim
1740 }
```

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).

```
1741 \hbox_set:Nw \l_@@_the_array_box
1742 \bool_if:NT \c_@@_testphase_table_bool
1743 { \UseTaggingSocket { tbl / hmode / begin } }
1744 \skip_horizontal:N \l_@@_left_margin_dim
1745 \skip_horizontal:N \l_@@_extra_left_margin_dim
1746 \c_math_toggle_token
1747 \bool_if:NTF \l_@@_light_syntax_bool
1748 { \use:c { @@-light-syntax } }
1749 { \use:c { @@-normal-syntax } }
1750 }
```

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

```
1751 \cs_new_protected_nopar:Npn \@@_CodeBefore_Body:w #1 \Body
1752 {
1753 \tl_set:Nn \l_tmpa_tl { #1 }
1754 \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
1755 { \@@_rescan_for_spanish:N \l_tmpa_tl }
1756 \tl_gput_left:No \g_@@_pre_code_before_tl \l_tmpa_tl
1757 \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...

```
1758 \@@_pre_array:
1759 }
```

10 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).

```
1760 \cs_new_protected:Npn \@@_pre_code_before:
1761 {
```

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).

```
1762 \int_set:Nn \c{iRow} { \seq_item:Nn \g_@@_size_seq 2 }
1763 \int_set:Nn \c{jCol} { \seq_item:Nn \g_@@_size_seq 5 }
1764 \int_set_eq:NN \g_@@_row_total_int { \seq_item:Nn \g_@@_size_seq 3 }
1765 \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.

```
1766 \pgfsys@markposition { \@@_env: - position }
1767 \pgfsys@getposition { \@@_env: - position } \@@_picture_position:
1768 \pgfpicture
1769 \pgf@relevantforpicturesizefalse
```

First, the recreation of the `row` nodes.

```
1770 \int_step_inline:nnn \l_@@_first_row_int { \g_@@_row_total_int + 1 }
1771 {
1772 \pgfsys@getposition { \@@_env: - row - ##1 } \@@_node_position:
1773 \pgfcoordinate { \@@_env: - row - ##1 }
1774 { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1775 }
```

Now, the recreation of the `col` nodes.

```
1776 \int_step_inline:nnn \l_@@_first_col_int { \g_@@_col_total_int + 1 }
1777 {
1778 \pgfsys@getposition { \@@_env: - col - ##1 } \@@_node_position:
1779 \pgfcoordinate { \@@_env: - col - ##1 }
1780 { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1781 }
```

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

```
1782 \@@_create_diag_nodes:
```

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

```
1783 \bool_if:NT \g_@@_recreate_cell_nodes_bool \@@_recreate_cell_nodes:
1784 \endpgfpicture
```

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

```
1785 \@@_create_blocks_nodes:
1786 \IfPackageLoadedT { tikz }
1787 {
1788 \tikzset
1789 {
1790 every-picture / .style =
1791 { overlay , name-prefix = \@@_env: - }
1792 }
1793 }
1794 \cs_set_eq:NN \cellcolor \@@_cellcolor
1795 \cs_set_eq:NN \rectanglecolor \@@_rectanglecolor
1796 \cs_set_eq:NN \roundedrectanglecolor \@@_roundedrectanglecolor
1797 \cs_set_eq:NN \rowcolor \@@_rowcolor
1798 \cs_set_eq:NN \rowcolors \@@_rowcolors
```

```

1799 \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors
1800 \cs_set_eq:NN \arraycolor \@@_arraycolor
1801 \cs_set_eq:NN \columncolor \@@_columncolor
1802 \cs_set_eq:NN \chessboardcolors \@@_chessboardcolors
1803 \cs_set_eq:NN \SubMatrix \@@_SubMatrix_in_code_before
1804 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
1805 \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
1806 }

```

```

1807 \cs_new_protected:Npn \@@_exec_code_before:
1808 {
1809   \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`.

```

1810 \@@_add_to_colors_seq:nn { { nocolor } } { }
1811 \bool_gset_false:N \g_@@_recreate_cell_nodes_bool
1812 \group_begin:

```

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

```

1813 \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`).

```

1814 \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
1815 { \@@_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`.

```

1816 \exp_last_unbraced:No \@@_CodeBefore_keys:
1817 \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.

```

1818 \@@_actually_color:
1819 \l_@@_code_before_tl
1820 \q_stop
1821 \bool_if:NT \l_@@_tabular_bool \c_math_toggle_token
1822 \group_end:
1823 \bool_if:NT \g_@@_recreate_cell_nodes_bool
1824 { \tl_put_left:Nn \@@_node_for_cell: \@@_patch_node_for_cell: }
1825 }

```

```

1826 \keys_define:nn { nicematrix / CodeBefore }
1827 {
1828   create-cell-nodes .bool_gset:N = \g_@@_recreate_cell_nodes_bool ,
1829   create-cell-nodes .default:n = true ,
1830   sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1831   sub-matrix .value_required:n = true ,
1832   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1833   delimiters / color .value_required:n = true ,
1834   unknown .code:n = \@@_error:n { Unknown~key~for~CodeBefore }
1835 }

```

```

1836 \NewDocumentCommand \@@_CodeBefore_keys: { 0 { } }
1837 {
1838   \keys_set:nn { nicematrix / CodeBefore } { #1 }
1839   \@@_CodeBefore:w
1840 }

```

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.

```

1841 \cs_new_protected:Npn \@@_CodeBefore:w #1 \q_stop
1842 {
1843   \bool_if:NT \g_@@_aux_found_bool
1844   {
1845     \@@_pre_code_before:
1846     #1
1847   }
1848 }

```

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.

```

1849 \cs_new_protected:Npn \@@_recreate_cell_nodes:
1850 {
1851   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
1852   {
1853     \pgfsys@getposition { \@@_env: - ##1 - base } \@@_node_position:
1854     \pgfcoordinate { \@@_env: - row - ##1 - base }
1855     { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1856     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
1857     {
1858       \cs_if_exist:cT
1859       { pgf @ sys @ pdf @ mark @ pos @ \@@_env: - ##1 - #####1 - NW }
1860       {
1861         \pgfsys@getposition
1862         { \@@_env: - ##1 - #####1 - NW }
1863         \@@_node_position:
1864         \pgfsys@getposition
1865         { \@@_env: - ##1 - #####1 - SE }
1866         \@@_node_position_i:
1867         \@@_pgf_rect_node:nnn
1868         { \@@_env: - ##1 - #####1 }
1869         { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1870         { \pgfpointdiff \@@_picture_position: \@@_node_position_i: }
1871       }
1872     }
1873   }
1874   \int_step_inline:nn \c@iRow
1875   {
1876     \pgfnodealias
1877     { \@@_env: - ##1 - last }
1878     { \@@_env: - ##1 - \int_use:N \c@jCol }
1879   }
1880   \int_step_inline:nn \c@jCol
1881   {
1882     \pgfnodealias
1883     { \@@_env: - last - ##1 }
1884     { \@@_env: - \int_use:N \c@iRow - ##1 }
1885   }
1886   \@@_create_extra_nodes:
1887 }

```

```

1888 \cs_new_protected:Npn \@@_create_blocks_nodes:
1889 {
1890   \pgfpicture
1891   \pgf@relevantforpicturesizefalse
1892   \pgfrememberpicturepositiononpagetrue
1893   \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
1894   { \@@_create_one_block_node:nnnnn ##1 }
1895   \endpgfpicture
1896 }

```

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

```

1897 \cs_new_protected:Npn \@@_create_one_block_node:nnnnn #1 #2 #3 #4 #5
1898 {
1899   \tl_if_empty:nF { #5 }
1900   {
1901     \@@_qpoint:n { col - #2 }
1902     \dim_set_eq:NN \l_tmpa_dim \pgf@x
1903     \@@_qpoint:n { #1 }
1904     \dim_set_eq:NN \l_tmpb_dim \pgf@y
1905     \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
1906     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
1907     \@@_qpoint:n { \int_eval:n { #3 + 1 } }
1908     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
1909     \@@_pgf_rect_node:nnnnn
1910     { \@@_env: - #5 }
1911     { \dim_use:N \l_tmpa_dim }
1912     { \dim_use:N \l_tmpb_dim }
1913     { \dim_use:N \l_@@_tmpc_dim }
1914     { \dim_use:N \l_@@_tmpd_dim }
1915   }
1916 }

1917 \cs_new_protected:Npn \@@_patch_for_revtext:
1918 {
1919   \cs_set_eq:NN \@addamp \@addamp@LaTeX
1920   \cs_set_eq:NN \insert@column \insert@column@array
1921   \cs_set_eq:NN \@classx \@classx@array
1922   \cs_set_eq:NN \@xarraycr \@xarraycr@array
1923   \cs_set_eq:NN \@arraycr \@arraycr@array
1924   \cs_set_eq:NN \@xargarraycr \@xargarraycr@array
1925   \cs_set_eq:NN \array \array@array
1926   \cs_set_eq:NN \@array \@array@array
1927   \cs_set_eq:NN \@tabular \@tabular@array
1928   \cs_set_eq:NN \@mkpream \@mkpream@array
1929   \cs_set_eq:NN \endarray \endarray@array
1930   \cs_set:Npn \@tabarray { \ifnextchar [ { \@array } { \@array [ c ] } }
1931   \cs_set:Npn \endtabular { \endarray $\egroup} % $
1932 }

```

11 The environment `{NiceArrayWithDelims}`

```

1933 \NewDocumentEnvironment { NiceArrayWithDelims }
1934 { m m 0 { } m ! 0 { } t \CodeBefore }
1935 {

```

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).

```

1936 \bool_if:NT \c_@@_revtex_bool \@@_patch_for_revtex:
1937 \@@_provide_pgfsyspdfmark:
1938 \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.

```

1939 \bgroup

1940 \tl_gset:Nn \g_@@_left_delim_tl { #1 }
1941 \tl_gset:Nn \g_@@_right_delim_tl { #2 }
1942 \tl_gset:Nn \g_@@_user_preamble_tl { #4 }
1943 \tl_if_empty:NT \g_@@_user_preamble_tl { \@@_fatal:n { empty-preamble } }

1944 \int_gzero:N \g_@@_block_box_int
1945 \dim_zero:N \g_@@_width_last_col_dim
1946 \dim_zero:N \g_@@_width_first_col_dim
1947 \bool_gset_false:N \g_@@_row_of_col_done_bool
1948 \str_if_empty:NT \g_@@_name_env_str
1949 { \str_gset:Nn \g_@@_name_env_str { NiceArrayWithDelims } }
1950 \bool_if:NTF \l_@@_tabular_bool
1951 \mode_leave_vertical:
1952 \@@_test_if_math_mode:
1953 \bool_if:NT \l_@@_in_env_bool { \@@_fatal:n { Yet~in~env } }
1954 \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.

```

1955 \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.

```

1956 \cs_if_exist:NT \tikz@library@external@loaded
1957 {
1958 \tikzexternaldisable
1959 \cs_if_exist:NT \ifstandalone
1960 { \tikzset { external / optimize = false } }
1961 }

```

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

```

1962 \int_gincr:N \g_@@_env_int
1963 \bool_if:NF \l_@@_block_auto_columns_width_bool
1964 { \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`).

```

1965 \seq_gclear:N \g_@@_blocks_seq
1966 \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`.

```

1967 \seq_gclear:N \g_@@_pos_of_stroken_blocks_seq
1968 \seq_gclear:N \g_@@_pos_of_xdots_seq
1969 \tl_gclear_new:N \g_@@_code_before_tl
1970 \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.

⁷e.g. `\color[rgb]{0.5,0.5,0}`

```

1971 \tl_if_exist:cTF { c_@@ _ \int_use:N \g_@@_env_int _ tl }
1972 {
1973   \bool_gset_true:N \g_@@_aux_found_bool
1974   \use:c { c_@@ _ \int_use:N \g_@@_env_int _ tl }
1975 }
1976 { \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.

```

1977 \tl_gclear:N \g_@@_aux_tl
1978 \tl_if_empty:NF \g_@@_code_before_tl
1979 {
1980   \bool_set_true:N \l_@@_code_before_bool
1981   \tl_put_right:No \l_@@_code_before_tl \g_@@_code_before_tl
1982 }
1983 \tl_if_empty:NF \g_@@_pre_code_before_tl
1984 { \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.

```

1985 \bool_if:NTF \g_@@_delims_bool
1986 { \keys_set:nn { nicematrix / pNiceArray } }
1987 { \keys_set:nn { nicematrix / NiceArray } }
1988 { #3 , #5 }

```

```

1989 \@@_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:.

```

1990 \bool_if:nTF { #6 } \@@_CodeBefore_Body:w \@@_pre_array:
1991 }

```

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

```

1992 {
1993   \bool_if:NTF \l_@@_light_syntax_bool
1994   { \use:c { end @@-light-syntax } }
1995   { \use:c { end @@-normal-syntax } }
1996   \c_math_toggle_token
1997   \skip_horizontal:N \l_@@_right_margin_dim
1998   \skip_horizontal:N \l_@@_extra_right_margin_dim
1999
2000   % awful workaround
2001   \int_compare:nNnT \g_@@_col_total_int = \c_one_int
2002   {
2003     \dim_compare:nNnT \l_@@_columns_width_dim > \c_zero_dim
2004     {
2005       \skip_horizontal:N - \l_@@_columns_width_dim
2006       \bool_if:NTF \l_@@_tabular_bool
2007       { \skip_horizontal:n { - 2 \tabcolsep } }
2008       { \skip_horizontal:n { - 2 \arraycolsep } }
2009     }
2010   }
2011   \hbox_set_end:

```

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

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

```

2012 \bool_if:NT \l_@@_width_used_bool
2013 {

```

```

2014 \int_if_zero:nT \g_@@_total_X_weight_int
2015 { \@@_error_or_warning:n { width-without-X-columns } }
2016 }

```

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 .

```

2017 \int_compare:nNnT \g_@@_total_X_weight_int > \c_zero_int
2018 {
2019   \tl_gput_right:Ne \g_@@_aux_tl
2020   {
2021     \bool_set_true:N \l_@@_X_columns_aux_bool
2022     \dim_set:Nn \l_@@_X_columns_dim
2023     {
2024       \dim_compare:nNnTF
2025       {
2026         \dim_abs:n
2027         { \l_@@_width_dim - \box_wd:N \l_@@_the_array_box }
2028       }
2029       <
2030       { 0.001 pt }
2031       { \dim_use:N \l_@@_X_columns_dim }
2032       {
2033         \dim_eval:n
2034         {
2035           ( \l_@@_width_dim - \box_wd:N \l_@@_the_array_box )
2036           / \int_use:N \g_@@_total_X_weight_int
2037           + \l_@@_X_columns_dim
2038         }
2039       }
2040     }
2041   }
2042 }

```

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).

```

2043 \int_compare:nNnT \l_@@_last_row_int > { -2 }
2044 {
2045   \bool_if:NF \l_@@_last_row_without_value_bool
2046   {
2047     \int_compare:nNnF \l_@@_last_row_int = \c@iRow
2048     {
2049       \@@_error:n { Wrong-last-row }
2050       \int_gset_eq:NN \l_@@_last_row_int \c@iRow
2051     }
2052   }
2053 }

```

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

```

2054 \int_gset_eq:NN \c@jCol \g_@@_col_total_int
2055 \bool_if:NNT \g_@@_last_col_found_bool
2056 { \int_gdecr:N \c@jCol }
2057 {
2058   \int_compare:nNnT \l_@@_last_col_int > { -1 }
2059   { \@@_error:n { last-col-not-used } }
2060 }

```

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

```

2061 \int_gset_eq:NN \g_@@_row_total_int \c@iRow

```

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


```
2062 \int_compare:nNt \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. 90).

```
2063 \int_if_zero:nT \l_@@_first_col_int
2064 { \skip_horizontal:N \g_@@_width_first_col_dim }
```

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

```
2065 \bool_if:nTF { ! \g_@@_delims_bool }
2066 {
2067   \tl_if_eq:NNTF \l_@@_baseline_tl \c_@@_c_tl
2068   \@@_use_arraybox_with_notes_c:
2069   {
2070     \tl_if_eq:NNTF \l_@@_baseline_tl \c_@@_b_tl
2071     \@@_use_arraybox_with_notes_b:
2072     \@@_use_arraybox_with_notes:
2073   }
2074 }
```

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).

```
2075 {
2076   \int_if_zero:nTF \l_@@_first_row_int
2077   {
2078     \dim_set_eq:NN \l_tmpa_dim \g_@@_dp_row_zero_dim
2079     \dim_add:Nn \l_tmpa_dim \g_@@_ht_row_zero_dim
2080   }
2081   { \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

```
2082 \int_compare:nNtTF \l_@@_last_row_int > { -2 }
2083 {
2084   \dim_set_eq:NN \l_tmpb_dim \g_@@_ht_last_row_dim
2085   \dim_add:Nn \l_tmpb_dim \g_@@_dp_last_row_dim
2086 }
2087 { \dim_zero:N \l_tmpb_dim }
2088 \hbox_set:Nn \l_tmpa_box
2089 {
2090   \c_math_toggle_token
2091   \@@_color:o \l_@@_delimiters_color_tl
2092   \exp_after:wN \left \g_@@_left_delim_tl
2093   \vcenter
2094   {
```

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.

```
2095   \skip_vertical:n { -\l_tmpa_dim - \arrayrulewidth }
2096   \hbox
2097   {
2098     \bool_if:NtF \l_@@_tabular_bool
2099     { \skip_horizontal:N -\tabcolsep }
2100     { \skip_horizontal:N -\arraycolsep }
2101     \@@_use_arraybox_with_notes_c:
2102     \bool_if:NtF \l_@@_tabular_bool
2103     { \skip_horizontal:N -\tabcolsep }
2104     { \skip_horizontal:N -\arraycolsep }
2105   }
```

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

```
2106   \skip_vertical:N -\l_tmpb_dim
```

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).

```

2107         \skip_vertical:N \arrayrulewidth
2108     }
2109     \exp_after:wN \right \g_@@_right_delim_tl
2110     \c_math_toggle_token
2111 }

```

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.

```

2112     \bool_if:NTF \l_@@_delimiters_max_width_bool
2113     {
2114         \@@_put_box_in_flow_bis:nn
2115         \g_@@_left_delim_tl
2116         \g_@@_right_delim_tl
2117     }
2118     \@@_put_box_in_flow:
2119 }

```

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. 91).

```

2120     \bool_if:NT \g_@@_last_col_found_bool
2121     { \skip_horizontal:N \g_@@_width_last_col_dim }
2122     \bool_if:NT \l_@@_preamble_bool
2123     {
2124         \int_compare:nNnT \c@jCol < \g_@@_static_num_of_col_int
2125         { \@@_warning_gredirect_none:n { columns-not-used } }
2126     }
2127     \@@_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.

```

2128     \egroup

```

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

```

2129     \iow_now:Nn \@mainaux { \ExplSyntaxOn }
2130     \iow_now:Nn \@mainaux { \char_set_catcode_space:n { 32 } }
2131     \iow_now:Ne \@mainaux
2132     {
2133         \tl_gset:cn { c_@@_ \int_use:N \g_@@_env_int _ tl }
2134         { \exp_not:o \g_@@_aux_tl }
2135     }
2136     \iow_now:Nn \@mainaux { \ExplSyntaxOff }

2137     \bool_if:NT \g_@@_footnote_bool \endsavenotes
2138 }

```

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

12 We construct the preamble of the array

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

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

```

2139 \cs_new_protected:Npn \@@_transform_preamble:
2140 {
2141     \@@_transform_preamble_i:
2142     \@@_transform_preamble_ii:
2143 }

```

```

2144 \cs_new_protected:Npn \@@_transform_preamble_i:
2145 {
2146   \int_gzero:N \c@jCol

```

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

```

2147   \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.

```

2148   \bool_gset_false:N \g_tmpb_bool

```

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

```

2149   \tl_gclear_new:N \g_@@_pre_cell_tl

```

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

```

2150   \int_zero:N \l_tmpa_int
2151   \tl_gclear:N \g_@@_array_preamble_tl
2152   \tl_if_eq:NNTF \l_@@_vlines_clist \c_@@_all_tl
2153   {
2154     \tl_gset:Nn \g_@@_array_preamble_tl
2155       { ! { \skip_horizontal:N \arrayrulewidth } }
2156   }
2157   {
2158     \clist_if_in:NnT \l_@@_vlines_clist 1
2159     {
2160       \tl_gset:Nn \g_@@_array_preamble_tl
2161         { ! { \skip_horizontal:N \arrayrulewidth } }
2162     }
2163   }

```

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

```

2164   \exp_last_unbraced:No \@@_rec_preamble:n \g_@@_user_preamble_tl \@@_stop:
2165   \int_gset_eq:NN \g_@@_static_num_of_col_int \c@jCol

2166   \@@_replace_columncolor:
2167 }

```

```

2168 \hook_gput_code:nnn { begindocument } { . }
2169 {
2170   \IfPackageLoadedTF { colortbl }
2171   {

```

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).

```

2172     \regex_const:Nn \c_@@_columncolor_regex { \c { columncolor } }
2173     \cs_new_protected:Npn \@@_replace_columncolor:
2174     {
2175       \regex_replace_all:NnN
2176         \c_@@_columncolor_regex
2177         { \c { @@_columncolor_preamble } }
2178       \g_@@_array_preamble_tl
2179     }
2180   }
2181   {
2182     \cs_new_protected:Npn \@@_replace_columncolor:
2183     { \cs_set_eq:NN \columncolor \@@_columncolor_preamble }
2184   }
2185 }

```

```

2186 \cs_new_protected:Npn \@@_transform_preamble_ii:
2187 {

```

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

```

2188 \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2189 {
2190   \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2191   { \bool_gset_true:N \g_@@_delims_bool }
2192 }
2193 { \bool_gset_true:N \g_@@_delims_bool }

```

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

```

2194 \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).

```

2195 \int_if_zero:nTF \l_@@_first_col_int
2196 { \tl_gput_left:No \g_@@_array_preamble_tl \c_@@_preamble_first_col_tl }
2197 {
2198   \bool_if:NF \g_@@_delims_bool
2199   {
2200     \bool_if:NF \l_@@_tabular_bool
2201     {
2202       \tl_if_empty:NT \l_@@_vlines_clist
2203       {
2204         \bool_if:NF \l_@@_exterior_arraycolsep_bool
2205         { \tl_gput_left:Nn \g_@@_array_preamble_tl { @ { } } }
2206       }
2207     }
2208   }
2209 }
2210 \int_compare:nNnTF \l_@@_last_col_int > { -1 }
2211 { \tl_gput_right:No \g_@@_array_preamble_tl \c_@@_preamble_last_col_tl }
2212 {
2213   \bool_if:NF \g_@@_delims_bool
2214   {
2215     \bool_if:NF \l_@@_tabular_bool
2216     {
2217       \tl_if_empty:NT \l_@@_vlines_clist
2218       {
2219         \bool_if:NF \l_@@_exterior_arraycolsep_bool
2220         { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { } } }
2221       }
2222     }
2223   }
2224 }

```

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`).

```

2225 \dim_compare:nNnT \l_@@_tabular_width_dim = \c_zero_dim
2226 {
2227   \tl_gput_right:Nn \g_@@_array_preamble_tl
2228   { > { \@@_error_too_much_cols: } 1 }
2229 }
2230 }

```

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.

```

2231 \cs_new_protected:Npn \@@_rec_preamble:n #1
2232 {

```

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

```
2233 \cs_if_exist:cTF { @@ _ \token_to_str:N #1 }
2234 { \use:c { @@ _ \token_to_str:N #1 } { #1 } }
2235 {
```

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

```
2236 \cs_if_exist:cTF { NC @ find @ #1 }
2237 {
2238 \tl_set_eq:Nc \l_tmpb_tl { NC @ rewrite @ #1 }
2239 \exp_last_unbraced:No \@@_rec_preamble:n \l_tmpb_tl
2240 }
2241 {
2242 \tl_if_eq:nnT { #1 } { S }
2243 { \@@_fatal:n { unknown~column~type~S } }
2244 { \@@_fatal:nn { unknown~column~type } { #1 } }
2245 }
2246 }
2247 }
```

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

```
2248 \cs_new:Npn \@@_c #1
2249 {
2250 \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2251 \tl_gclear:N \g_@@_pre_cell_tl
2252 \tl_gput_right:Nn \g_@@_array_preamble_tl
2253 { > \@@_cell_begin:w c < \@@_cell_end: }
```

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

```
2254 \int_gincr:N \c@jCol
2255 \@@_rec_preamble_after_col:n
2256 }

2257 \cs_new:Npn \@@_l #1
2258 {
2259 \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2260 \tl_gclear:N \g_@@_pre_cell_tl
2261 \tl_gput_right:Nn \g_@@_array_preamble_tl
2262 {
2263 > { \@@_cell_begin:w \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_l_tl }
2264 l
2265 < \@@_cell_end:
2266 }
2267 \int_gincr:N \c@jCol
2268 \@@_rec_preamble_after_col:n
2269 }

2270 \cs_new:Npn \@@_r #1
2271 {
2272 \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2273 \tl_gclear:N \g_@@_pre_cell_tl
2274 \tl_gput_right:Nn \g_@@_array_preamble_tl
2275 {
2276 > { \@@_cell_begin:w \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_r_tl }
2277 r
2278 < \@@_cell_end:
2279 }
2280 \int_gincr:N \c@jCol
2281 \@@_rec_preamble_after_col:n
2282 }
```

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`.

For ! and @

```

2283 \cs_new:cpn { @@ _ \token_to_str:N ! } #1 #2
2284 {
2285   \tl_gput_right:Nn \g_@@_array_preamble_tl { #1 { #2 } }
2286   \@@_rec_preamble:n
2287 }
2288 \cs_set_eq:cc { @@ _ \token_to_str:N @ } { @@ _ \token_to_str:N ! }

```

For |

```

2289 \cs_new:cpn { @@ _ | } #1
2290 {

```

\l_tmpa_int is the number of successive occurrences of |

```

2291   \int_incr:N \l_tmpa_int
2292   \@@_make_preamble_i_i:n
2293 }
2294 \cs_new_protected:Npn \@@_make_preamble_i_i:n #1
2295 {
2296   \str_if_eq:nnTF { #1 } |
2297   { \use:c { @@ _ | } | }
2298   { \@@_make_preamble_i_ii:nn { } #1 }
2299 }
2300 \cs_new_protected:Npn \@@_make_preamble_i_ii:nn #1 #2
2301 {
2302   \str_if_eq:nnTF { #2 } [
2303   { \@@_make_preamble_i_iii:nn { #1 } [ ] }
2304   { \@@_make_preamble_i_iii:nn { #2 } { #1 } }
2305 ]
2306 \cs_new_protected:Npn \@@_make_preamble_i_ii:nw #1 [ #2 ]
2307 { \@@_make_preamble_i_ii:nn { #1 , #2 } }
2308 \cs_new_protected:Npn \@@_make_preamble_i_iii:nn #1 #2
2309 {
2310   \@@_compute_rule_width:n { multiplicity = \l_tmpa_int , #2 }
2311   \tl_gput_right:Ne \g_@@_array_preamble_tl
2312   {

```

Here, the command \dim_eval:n is mandatory.

```

2313   \exp_not:N ! { \skip_horizontal:n { \dim_eval:n { \l_@@_rule_width_dim } } }
2314 }
2315 \tl_gput_right:Ne \g_@@_pre_code_after_tl
2316 {
2317   \@@_vline:n
2318   {
2319     position = \int_eval:n { \c@jCol + 1 } ,
2320     multiplicity = \int_use:N \l_tmpa_int ,
2321     total-width = \dim_use:N \l_@@_rule_width_dim ,
2322     #2
2323   }

```

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.

```

2324   }
2325   \int_zero:N \l_tmpa_int
2326   \str_if_eq:nnT { #1 } { \@@_stop: } { \bool_gset_true:N \g_tmpb_bool }
2327   \@@_rec_preamble:n #1
2328 }
2329 \cs_new:cpn { @@ _ > } #1 #2
2330 {
2331   \tl_gput_right:Nn \g_@@_pre_cell_tl { > { #2 } }
2332   \@@_rec_preamble:n
2333 }

```

```
2334 \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.

```
2335 \keys_define:nn { nicematrix / p-column }
2336 {
2337   r .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_r_str ,
2338   r .value_forbidden:n = true ,
2339   c .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_c_str ,
2340   c .value_forbidden:n = true ,
2341   l .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_l_str ,
2342   l .value_forbidden:n = true ,
2343   R .code:n =
2344     \IfPackageLoadedTF { ragged2e }
2345     { \str_set_eq:NN \l_@@_hpos_col_str \c_@@_R_str }
2346     {
2347       \@@_error_or_warning:n { ragged2e~not~loaded }
2348       \str_set_eq:NN \l_@@_hpos_col_str \c_@@_r_str
2349     } ,
2350   R .value_forbidden:n = true ,
2351   L .code:n =
2352     \IfPackageLoadedTF { ragged2e }
2353     { \str_set_eq:NN \l_@@_hpos_col_str \c_@@_L_str }
2354     {
2355       \@@_error_or_warning:n { ragged2e~not~loaded }
2356       \str_set_eq:NN \l_@@_hpos_col_str \c_@@_l_str
2357     } ,
2358   L .value_forbidden:n = true ,
2359   C .code:n =
2360     \IfPackageLoadedTF { ragged2e }
2361     { \str_set_eq:NN \l_@@_hpos_col_str \c_@@_C_str }
2362     {
2363       \@@_error_or_warning:n { ragged2e~not~loaded }
2364       \str_set_eq:NN \l_@@_hpos_col_str \c_@@_c_str
2365     } ,
2366   C .value_forbidden:n = true ,
2367   S .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_si_str ,
2368   S .value_forbidden:n = true ,
2369   p .code:n = \str_set:Nn \l_@@_vpos_col_str { p } ,
2370   p .value_forbidden:n = true ,
2371   t .meta:n = p ,
2372   m .code:n = \str_set:Nn \l_@@_vpos_col_str { m } ,
2373   m .value_forbidden:n = true ,
2374   b .code:n = \str_set:Nn \l_@@_vpos_col_str { b } ,
2375   b .value_forbidden:n = true ,
2376 }
```

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

```
2377 \cs_new:Npn \@@_p #1
2378 {
2379   \str_set:Nn \l_@@_vpos_col_str { #1 }
```

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

```
2380 \@@_make_preamble_ii_i:n
2381 }
2382 \cs_set_eq:NN \@@_b \@@_p
2383 \cs_set_eq:NN \@@_m \@@_p
2384 \cs_new_protected:Npn \@@_make_preamble_ii_i:n #1
2385 {
2386   \str_if_eq:nnTF { #1 } { [ ]
2387     { \@@_make_preamble_ii_ii:w [ ]
2388       { \@@_make_preamble_ii_ii:w [ ] { #1 } }
2389     }
```

```

2390 \cs_new_protected:Npn \@@_make_preamble_ii_ii:w [ #1 ]
2391 { \@@_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.

```

2392 \cs_new_protected:Npn \@@_make_preamble_ii_iii:nn #1 #2
2393 {

```

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).

```

2394   \str_set_eq:NN \l_@@_hpos_col_str \c_@@_j_str
2395   \@@_keys_p_column:n { #1 }
2396   \@@_make_preamble_ii_iv:nnn { #2 } { minipage } { }
2397 }

2398 \cs_new_protected:Npn \@@_keys_p_column:n #1
2399 { \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.

```

2400 \cs_new_protected:Npn \@@_make_preamble_ii_iv:nnn #1 #2 #3
2401 {
2402   \use:e
2403   {
2404     \@@_make_preamble_ii_v:nnnnnnnn
2405     { \str_if_eq:onTF \l_@@_vpos_col_str { p } { t } { b } }
2406     { \dim_eval:n { #1 } }
2407   }

```

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.

```

2408   \str_if_eq:NNTF \l_@@_hpos_col_str \c_@@_j_str
2409   { \tl_clear:N \exp_not:N \l_@@_hpos_cell_tl }
2410   {
2411     \cs_set_nopar:Npn \exp_not:N \l_@@_hpos_cell_tl
2412     { \str_lowercase:o \l_@@_hpos_col_str }
2413   }
2414   \str_case:on \l_@@_hpos_col_str
2415   {
2416     c { \exp_not:N \centering }
2417     l { \exp_not:N \raggedright }
2418     r { \exp_not:N \raggedleft }
2419     C { \exp_not:N \Centering }
2420     L { \exp_not:N \RaggedRight }
2421     R { \exp_not:N \RaggedLeft }
2422   }
2423   #3
2424 }
2425 { \str_if_eq:onT \l_@@_vpos_col_str { m } \@@_center_cell_box: }
2426 { \str_if_eq:onT \l_@@_hpos_col_str { si } \siunitx_cell_begin:w }
2427 { \str_if_eq:onT \l_@@_hpos_col_str { si } \siunitx_cell_end: }
2428 { #2 }
2429 {
2430   \str_case:onF \l_@@_hpos_col_str
2431   {
2432     { j } { c }
2433     { si } { c }
2434   }

```

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

```

2435     { \str_lowercase:o \l_@@_hpos_col_str }
2436   }
2437 }

```


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

```
2438 \int_gincr:N \c@jCol
2439 \@@_rec_preamble_after_col:n
2440 }
```

#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*.

```
2441 \cs_new_protected:Npn \@@_make_preamble_ii_v:nnnnnnnn #1 #2 #3 #4 #5 #6 #7 #8
2442 {
2443   \tl_if_eq:NNTF \l_@@_hpos_col_str \c_@@_si_str
2444   {
2445     \tl_gput_right:Nn \g_@@_array_preamble_tl
2446     { > { \@@_test_if_empty_for_S: } }
2447   }
2448   {
2449     \tl_gput_right:Nn \g_@@_array_preamble_tl
2450     { > { \@@_test_if_empty: } }
2451   }
2452   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2453   \tl_gc_clear:N \g_@@_pre_cell_tl
2454   \tl_gput_right:Nn \g_@@_array_preamble_tl
2455   {
2456     > {
```

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.

```
2457 \dim_set:Nn \l_@@_col_width_dim { #2 }
2458 \bool_if:NT \c_@@_testphase_table_bool
2459 { \tag_struct_begin:n { tag = Div } }
2460 \@@_cell_begin:w
```

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

```
2461 \use:c { #7 } [ #1 ] { #2 }
```

The following lines have been taken from array.sty.

```
2462 \everypar
2463 {
2464   \vrule height \box_ht:N \@arstrutbox width \c_zero_dim
2465   \everypar { }
2466 }
2467 \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.).

```
2468 #3
```

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

```
2469 \g_@@_row_style_tl
2470 \arraybackslash
2471 #5
2472 }
2473 #8
```

```

2474 < {
2475     #6

```

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

```

2476     \@finalstrut \@arstrutbox
2477     \use:c { end #7 }

```

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

```

2478     #4
2479     \@@_cell_end:
2480     \bool_if:NT \c_@@_testphase_table_bool \tag_struct_end:
2481 }
2482 }
2483 }

```

```

2484 \str_new:N \c_@@_ignorespaces_str
2485 \str_set:Ne \c_@@_ignorespaces_str { \ignorespaces }
2486 \str_remove_all:Nn \c_@@_ignorespaces_str { ~ }
2487 \cs_new_protected:Npn \@@_test_if_empty:
2488 { \peek_after:Nw \@@_test_if_empty_i: }
2489 \cs_new_protected:Npn \@@_test_if_empty_i:
2490 {
2491     \str_set:Ne \l_tmpa_str { \token_to_meaning:N \l_peek_token }
2492     \str_if_eq:NNT \l_tmpa_str \c_@@_ignorespaces_str
2493     { \@@_test_if_empty:w }
2494 }
2495 \cs_new_protected:Npn \@@_test_if_empty:w \ignorespaces
2496 { \peek_after:Nw \@@_test_if_empty_ii: }

```

```

2497 \cs_new_protected:Npn \@@_nullify_cell:
2498 {
2499     \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2500     {
2501         \box_set_wd:Nn \l_@@_cell_box \c_zero_dim
2502         \skip_horizontal:N \l_@@_col_width_dim
2503     }
2504 }

```

```

2505 \bool_if:NNT \c_@@_tagging_array_bool
2506 {
2507     \cs_new_protected:Npn \@@_test_if_empty_ii:
2508     { \peek_meaning:NT \textonly@unskip \@@_nullify_cell: }
2509 }

```

In the old version of `array`, we test whether it begins by `\ignorespaces\unskip`. However, in some circumstances, for example when `\collectcell` of `colcell` is used, the cell does not begin with `\ignorespaces`. In that case, we consider as not empty... First, we test if the next token is `\ignorespaces` and it's not very easy...

```

2510 {
2511     \cs_new_protected:Npn \@@_test_if_empty_ii:
2512     { \peek_meaning:NT \unskip \@@_nullify_cell: }
2513 }
2514 \cs_new_protected:Npn \@@_test_if_empty_for_S:
2515 {
2516     \peek_meaning:NT \__siunitx_table_skip:n
2517     {
2518         \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2519         { \box_set_wd:Nn \l_@@_cell_box \c_zero_dim }
2520     }
2521 }

```

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

```
2522 \cs_new_protected:Npn \l_@@_center_cell_box:
2523 {
```

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

```
2524 \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2525 {
2526   \int_compare:nNnT
2527     { \box_ht:N \l_@@_cell_box }
2528     >
```

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).

```
2529     { \box_ht:N \strutbox }
2530     {
2531       \hbox_set:Nn \l_@@_cell_box
2532       {
2533         \box_move_down:nn
2534         {
2535           ( \box_ht:N \l_@@_cell_box - \box_ht:N \@arstrutbox
2536             + \baselineskip ) / 2
2537         }
2538         { \box_use:N \l_@@_cell_box }
2539       }
2540     }
2541   }
2542 }
```

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

```
2543 \cs_new:Npn \l_@@_V #1 #2
2544 {
2545   \str_if_eq:nnTF { #2 } { [ ] }
2546     { \@@_make_preamble_V_i:w [ ] }
2547     { \@@_make_preamble_V_i:w [ ] { #2 } }
2548 }
2549 \cs_new_protected:Npn \@@_make_preamble_V_i:w [ #1 ]
2550 { \@@_make_preamble_V_ii:nn { #1 } }
2551 \cs_new_protected:Npn \@@_make_preamble_V_ii:nn #1 #2
2552 {
2553   \str_set:Nn \l_@@_vpos_col_str { p }
2554   \str_set_eq:NN \l_@@_hpos_col_str \c_@@_j_str
2555   \@@_keys_p_column:n { #1 }
2556   \IfPackageLoadedTF { varwidth }
2557     { \@@_make_preamble_ii_iv:nnn { #2 } { varwidth } { } }
2558     {
2559       \@@_error_or_warning:n { varwidth-not-loaded }
2560       \@@_make_preamble_ii_iv:nnn { #2 } { minipage } { }
2561     }
2562 }
```

For `w` and `W`

```
2563 \cs_new:Npn \@@_w { \@@_make_preamble_w:nnnn { } }
2564 \cs_new:Npn \@@_W { \@@_make_preamble_w:nnnn { \@@_special_W: } }
```

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

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

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

#4 is the width of the column.

```

2565 \cs_new_protected:Npn \@@_make_preamble_w:nnnn #1 #2 #3 #4
2566 {
2567   \str_if_eq:nnTF { #3 } { s }
2568   { \@@_make_preamble_w_i:nnnn { #1 } { #4 } }
2569   { \@@_make_preamble_w_ii:nnnn { #1 } { #2 } { #3 } { #4 } }
2570 }

```

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.

```

2571 \cs_new_protected:Npn \@@_make_preamble_w_i:nnnn #1 #2
2572 {
2573   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2574   \tl_gclear:N \g_@@_pre_cell_tl
2575   \tl_gput_right:Nn \g_@@_array_preamble_tl
2576   {
2577     > {
2578       \dim_set:Nn \l_@@_col_width_dim { #2 }
2579       \@@_cell_begin:w
2580       \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_c_tl
2581     }
2582     c
2583     < {
2584       \@@_cell_end_for_w_s:
2585       #1
2586       \@@_adjust_size_box:
2587       \box_use_drop:N \l_@@_cell_box
2588     }
2589   }
2590   \int_gincr:N \c@jCol
2591   \@@_rec_preamble_after_col:n
2592 }

```

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

```

2593 \cs_new_protected:Npn \@@_make_preamble_w_ii:nnnn #1 #2 #3 #4
2594 {
2595   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2596   \tl_gclear:N \g_@@_pre_cell_tl
2597   \tl_gput_right:Nn \g_@@_array_preamble_tl
2598   {
2599     > {

```

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.

```

2600       \dim_set:Nn \l_@@_col_width_dim { #4 }
2601       \hbox_set:Nw \l_@@_cell_box
2602       \@@_cell_begin:w
2603       \cs_set_nopar:Npn \l_@@_hpos_cell_tl { #3 }
2604     }
2605     c
2606     < {
2607       \@@_cell_end:
2608       \hbox_set_end:
2609       #1
2610       \@@_adjust_size_box:
2611       \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2612     }
2613   }

```

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

```

2614 \int_gincr:N \c@jCol
2615 \@@_rec_preamble_after_col:n
2616 }

2617 \cs_new_protected:Npn \@@_special_W:
2618 {
2619 \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \l_@@_col_width_dim
2620 { \@@_warning:n { W~warning } }
2621 }

```

For S (of siunitx).

```

2622 \cs_new:Npn \@@_S #1 #2
2623 {
2624 \str_if_eq:nnTF { #2 } { [ ] }
2625 { \@@_make_preamble_S:w [ ] }
2626 { \@@_make_preamble_S:w [ ] { #2 } }
2627 }

2628 \cs_new_protected:Npn \@@_make_preamble_S:w [ #1 ]
2629 { \@@_make_preamble_S_i:n { #1 } }

2630 \cs_new_protected:Npn \@@_make_preamble_S_i:n #1
2631 {
2632 \IfPackageLoadedF { siunitx } { \@@_fatal:n { siunitx~not~loaded } }
2633 \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2634 \tl_gclear:N \g_@@_pre_cell_tl
2635 \tl_gput_right:Nn \g_@@_array_preamble_tl
2636 {
2637 > {
2638 \@@_cell_begin:w
2639 \keys_set:nn { siunitx } { #1 }
2640 \siunitx_cell_begin:w
2641 }
2642 c
2643 < { \siunitx_cell_end: \@@_cell_end: }
2644 }

```

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

```

2645 \int_gincr:N \c@jCol
2646 \@@_rec_preamble_after_col:n
2647 }

```

For (, [and \{.

```

2648 \cs_new:cpn { @@ _ \token_to_str:N ( } #1 #2
2649 {
2650 \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.

```

2651 \int_if_zero:nTF \c@jCol
2652 {
2653 \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2654 {

```

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

```

2655 \tl_gset:Nn \g_@@_left_delim_tl { #1 }
2656 \tl_gset_eq:NN \g_@@_right_delim_tl \c_@@_dot_tl
2657 \@@_rec_preamble:n #2
2658 }
2659 {
2660 \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2661 \@@_make_preamble_iv:nn { #1 } { #2 }
2662 }

```

```

2663     }
2664     { \@@_make_preamble_iv:nn { #1 } { #2 } }
2665   }
2666   \cs_set_eq:cc { @@ _ \token_to_str:N [ ] { @@ _ \token_to_str:N ( }
2667   \cs_set_eq:cc { @@ _ \token_to_str:N \{ } { @@ _ \token_to_str:N ( }
2668   \cs_new_protected:Npn \@@_make_preamble_iv:nn #1 #2
2669   {
2670     \tl_gput_right:Ne \g_@@_pre_code_after_tl
2671     { \@@_delimiter:nnn #1 { \int_eval:n { \c@jCol + 1 } } \c_true_bool }
2672     \tl_if_in:nnTF { ( [ \{ ) ] \} \left \right } { #2 }
2673     {
2674       \@@_error:nn { delimiter~after~opening } { #2 }
2675       \@@_rec_preamble:n
2676     }
2677     { \@@_rec_preamble:n #2 }
2678   }

```

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

```

2679 \cs_new:cpn { @@ _ \token_to_str:N \left } #1 { \use:c { @@ _ \token_to_str:N ( } }

```

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

```

2680 \cs_new:cpn { @@ _ \token_to_str:N ) } #1 #2
2681 {
2682   \bool_if:NT \l_@@_small_bool { \@@_fatal:n { Delimiter~with~small } }
2683   \tl_if_in:nnTF { ) ] \} } { #2 }
2684   { \@@_make_preamble_v:nnn #1 #2 }
2685   {
2686     \tl_if_eq:nnTF { \@@_stop: } { #2 }
2687     {
2688       \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2689       { \tl_gset:Nn \g_@@_right_delim_tl { #1 } }
2690       {
2691         \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2692         \tl_gput_right:Ne \g_@@_pre_code_after_tl
2693         { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2694         \@@_rec_preamble:n #2
2695       }
2696     }
2697     {
2698       \tl_if_in:nnT { ( [ \{ \left } { #2 }
2699       { \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } } }
2700       \tl_gput_right:Ne \g_@@_pre_code_after_tl
2701       { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2702       \@@_rec_preamble:n #2
2703     }
2704   }
2705 }
2706 \cs_set_eq:cc { @@ _ \token_to_str:N ] } { @@ _ \token_to_str:N ) }
2707 \cs_set_eq:cc { @@ _ \token_to_str:N \} } { @@ _ \token_to_str:N ) }
2708 \cs_new_protected:Npn \@@_make_preamble_v:nnn #1 #2 #3
2709 {
2710   \tl_if_eq:nnTF { \@@_stop: } { #3 }
2711   {
2712     \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2713     {
2714       \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2715       \tl_gput_right:Ne \g_@@_pre_code_after_tl
2716       { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2717       \tl_gset:Nn \g_@@_right_delim_tl { #2 }

```

```

2718     }
2719     {
2720         \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2721         \tl_gput_right:Ne \g_@@_pre_code_after_tl
2722         { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2723         \@@_error:nn { double~closing~delimiter } { #2 }
2724     }
2725 }
2726 {
2727     \tl_gput_right:Ne \g_@@_pre_code_after_tl
2728     { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2729     \@@_error:nn { double~closing~delimiter } { #2 }
2730     \@@_rec_preamble:n #3
2731 }
2732 }

2733 \cs_new:cpn { @@ _ \token_to_str:N \right } #1
2734 { \use:c { @@ _ \token_to_str:N ) } }

```

After a specifier of column, we have to test whether there is one or several $\langle \{ \dots \}$ because, after those potential $\langle \{ \dots \}$, we have to insert $!\{\backslash\text{skip_horizontal:N } \dots\}$ when the key `vlines` is used. In fact, we have also to test whether there is, after the $\langle \{ \dots \}$, a $\@ \{ \dots \}$.

```

2735 \cs_new_protected:Npn \@@_rec_preamble_after_col:n #1
2736 {
2737     \str_if_eq:nnTF { #1 } { < }
2738     \@@_rec_preamble_after_col_i:n
2739     {
2740         \str_if_eq:nnTF { #1 } { @ }
2741         \@@_rec_preamble_after_col_ii:n
2742         {
2743             \tl_if_eq:NNTF \l_@@_vlines_clist \c_@@_all_tl
2744             {
2745                 \tl_gput_right:Nn \g_@@_array_preamble_tl
2746                 { ! { \skip_horizontal:N \arrayrulewidth } }
2747             }
2748             {
2749                 \clist_if_in:NeT \l_@@_vlines_clist { \int_eval:n { \c@jCol + 1 } }
2750                 {
2751                     \tl_gput_right:Nn \g_@@_array_preamble_tl
2752                     { ! { \skip_horizontal:N \arrayrulewidth } }
2753                 }
2754             }
2755             \@@_rec_preamble:n { #1 }
2756         }
2757     }
2758 }

2759 \cs_new_protected:Npn \@@_rec_preamble_after_col_i:n #1
2760 {
2761     \tl_gput_right:Nn \g_@@_array_preamble_tl { < { #1 } }
2762     \@@_rec_preamble_after_col:n
2763 }

```

We have to catch a $\@ \{ \dots \}$ after a specifier of column because, if we have to draw a vertical rule, we have to add in that $\@ \{ \dots \}$ a $\backslash\text{hskip}$ corresponding to the width of the vertical rule.

```

2764 \cs_new_protected:Npn \@@_rec_preamble_after_col_ii:n #1
2765 {
2766     \tl_if_eq:NNTF \l_@@_vlines_clist \c_@@_all_tl
2767     {
2768         \tl_gput_right:Nn \g_@@_array_preamble_tl
2769         { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2770     }
2771     {

```

```

2772 \clist_if_in:N\neTF \l_@@_vlines_clist { \int_eval:n { \c@jCol + 1 } }
2773 {
2774 \tl_gput_right:Nn \g_@@_array_preamble_tl
2775 { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2776 }
2777 { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { #1 } } }
2778 }
2779 \@@_rec_preamble:n
2780 }

2781 \cs_new:cpn { @@ _ * } #1 #2 #3
2782 {
2783 \tl_clear:N \l_tmpa_tl
2784 \int_step_inline:nn { #2 } { \tl_put_right:Nn \l_tmpa_tl { #3 } }
2785 \exp_last_unbraced:No \@@_rec_preamble:n \l_tmpa_tl
2786 }

```

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

```

2787 \cs_new:cpn { @@ _ \token_to_str:N \NC@find } #1 { \@@_rec_preamble:n }

```

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

```

2788 \cs_new:Npn \@@_X #1 #2
2789 {
2790 \str_if_eq:nnTF { #2 } { [ ]
2791 { \@@_make_preamble_X:w [ ]
2792 { \@@_make_preamble_X:w [ ] #2 }
2793 }
2794 \cs_new_protected:Npn \@@_make_preamble_X:w [ #1 ]
2795 { \@@_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`).

```

2796 \keys_define:nn { nicematrix / X-column }
2797 { 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`.

```

2798 \cs_new_protected:Npn \@@_make_preamble_X:i:n #1
2799 {

```

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`).

```

2800 \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`).

```

2801 \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`.

```

2802 \int_zero_new:N \l_@@_weight_int
2803 \int_set_eq:NN \l_@@_weight_int \c_one_int
2804 \@@_keys_p_column:n { #1 }

```


The unknown keys are put in \l_tmpa_tl

```

2805 \keys_set:no { nicematrix / X-column } \l_tmpa_tl
2806 \int_compare:nNnT \l_@@_weight_int < \c_zero_int
2807 {
2808   \@@_error_or_warning:n { negative-weight }
2809   \int_set:Nn \l_@@_weight_int { - \l_@@_weight_int }
2810 }
2811 \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).

```

2812 \bool_if:NTF \l_@@_X_columns_aux_bool
2813 {
2814   \@@_make_preamble_ii_iv:nnn
2815   { \l_@@_weight_int \l_@@_X_columns_dim }
2816   { minipage }
2817   { \@@_no_update_width: }
2818 }
2819 {
2820   \tl_gput_right:Nn \g_@@_array_preamble_tl
2821   {
2822     > {
2823       \@@_cell_begin:w
2824       \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.

```

2825 \NotEmpty

```

The following code will nullify the box of the cell.

```

2826 \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2827 { \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.

```

2828 \begin { minipage } { 5 cm } \arraybackslash
2829 }
2830 c
2831 < {
2832   \end { minipage }
2833   \@@_cell_end:
2834 }
2835 }
2836 \int_gincr:N \c@jCol
2837 \@@_rec_preamble_after_col:n
2838 }
2839 }

2840 \cs_new_protected:Npn \@@_no_update_width:
2841 {
2842   \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2843   { \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing: }
2844 }

```

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

```

2845 \cs_new_protected:Npn \@@_make_preamble_vlism:n #1
2846 {
2847   \seq_gput_right:Ne \g_@@_cols_vlism_seq
2848   { \int_eval:n { \c@jCol + 1 } }
2849   \tl_gput_right:Ne \g_@@_array_preamble_tl
2850   { \exp_not:N ! { \skip_horizontal:N \arrayrulewidth } }
2851   \@@_rec_preamble:n
2852 }

```

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.

```
2853 \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).

```
2854 \cs_new_protected:cpn { @@ _ \token_to_str:N \hline }
2855 { \@@_fatal:n { Preamble-forgotten } }
2856 \cs_set_eq:cc { @@ _ \token_to_str:N \Hline } { @@ _ \token_to_str:N \hline }
2857 \cs_set_eq:cc { @@ _ \token_to_str:N \toprule } { @@ _ \token_to_str:N \hline }
2858 \cs_set_eq:cc { @@ _ \token_to_str:N \CodeBefore } { @@ _ \token_to_str:N \hline }
```

13 The redefinition of `\multicolumn`

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

```
2859 \cs_new:Npn \@@_multicolumn:nnn #1 #2 #3
2860 {
```

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`.

```
2861 \multispan { #1 }
2862 \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing:
2863 \begingroup
2864 \bool_if:NT \c_@@_testphase_table_bool
2865 { \tbl_update_multicolumn_cell_data:n { #1 } }
2866 \cs_set_nopar:Npn \@addamp
2867 { \legacy_if:nTF { @firstamp } { \@firstampfalse } { \@preamerr 5 } }
```

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

```
2868 \tl_gclear:N \g_@@_preamble_tl
2869 \@@_make_m_preamble:n #2 \q_stop
```

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

```
2870 \exp_args:No \@mkpream \g_@@_preamble_tl
2871 \@addtopreamble \@empty
2872 \endgroup
2873 \bool_if:NT \c_@@_testphase_table_bool
2874 { \UseTaggingSocket { tbl / colspan } { #1 } }
```

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

```
2875 \int_compare:nNnT { #1 } > \c_one_int
2876 {
2877 \seq_gput_left:Ne \g_@@_multicolumn_cells_seq
2878 { \int_use:N \c@iRow - \int_eval:n { \c@jCol + 1 } }
2879 \seq_gput_left:Nn \g_@@_multicolumn_sizes_seq { #1 }
2880 \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
2881 {
2882 {
2883 \int_if_zero:nTF \c@jCol
2884 { \int_eval:n { \c@iRow + 1 } }
2885 { \int_use:N \c@iRow }
2886 }
2887 { \int_eval:n { \c@jCol + 1 } }
2888 {
2889 \int_if_zero:nTF \c@jCol
2890 { \int_eval:n { \c@iRow + 1 } }
```

```

2891         { \int_use:N \c@iRow }
2892     }
2893     { \int_eval:n { \c@jCol + #1 } }
2894     { } % for the name of the block
2895 }
2896 }

```

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

```

2897 \RenewDocumentCommand \cellcolor { 0 { } m }
2898 {
2899     \@@_test_color_inside:
2900     \tl_gput_right:Nx \g_@@_pre_code_before_tl
2901     {
2902         \@@_rectanglecolor [ ##1 ]
2903         { \exp_not:n { ##2 } }
2904         { \int_use:N \c@iRow - \int_use:N \c@jCol }
2905         { \int_use:N \c@iRow - \int_eval:n { \c@jCol + #1 } }
2906     }
2907     \ignorespaces
2908 }

```

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

```

2909 \cs_set_nopar:Npn \@sharp { #3 }
2910 \@arstrut
2911 \@preamble
2912 \null

```

We add some lines.

```

2913 \int_gadd:Nn \c@jCol { #1 - 1 }
2914 \int_compare:nNnT \c@jCol > \g_@@_col_total_int
2915 { \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
2916 \ignorespaces
2917 }

```

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`.

```

2918 \cs_new_protected:Npn \@@_make_m_preamble:n #1
2919 {
2920     \str_case:nnF { #1 }
2921     {
2922         c { \@@_make_m_preamble_i:n #1 }
2923         l { \@@_make_m_preamble_i:n #1 }
2924         r { \@@_make_m_preamble_i:n #1 }
2925         > { \@@_make_m_preamble_ii:nn #1 }
2926         ! { \@@_make_m_preamble_ii:nn #1 }
2927         @ { \@@_make_m_preamble_ii:nn #1 }
2928         | { \@@_make_m_preamble_iii:n #1 }
2929         p { \@@_make_m_preamble_iv:nnn t #1 }
2930         m { \@@_make_m_preamble_iv:nnn c #1 }
2931         b { \@@_make_m_preamble_iv:nnn b #1 }
2932         w { \@@_make_m_preamble_v:nnnn { } #1 }
2933         W { \@@_make_m_preamble_v:nnnn { \@@_special_W: } #1 }
2934         \q_stop { }
2935     }
2936     {
2937         \cs_if_exist:cTF { NC @ find @ #1 }
2938         {
2939             \tl_set_eq:Nc \l_tmpa_tl { NC @ rewrite @ #1 }
2940             \exp_last_unbraced:No \@@_make_m_preamble:n \l_tmpa_tl
2941         }
2942         {

```

```

2943         \tl_if_eq:nnT { #1 } { S }
2944         { \@@_fatal:n { unknown~column~type~S } }
2945         { \@@_fatal:nn { unknown~column~type } { #1 } }
2946     }
2947 }
2948 }

```

For c, l and r

```

2949 \cs_new_protected:Npn \@@_make_m_preamble_i:n #1
2950 {
2951     \tl_gput_right:Nn \g_@@_preamble_tl
2952     {
2953         > { \@@_cell_begin:w \cs_set_nopar:Npn \l_@@_hpos_cell_tl { #1 } }
2954         #1
2955         < \@@_cell_end:
2956     }

```

We test for the presence of a <.

```

2957     \@@_make_m_preamble_x:n
2958 }

```

For >, ! and @

```

2959 \cs_new_protected:Npn \@@_make_m_preamble_ii:nn #1 #2
2960 {
2961     \tl_gput_right:Nn \g_@@_preamble_tl { #1 { #2 } }
2962     \@@_make_m_preamble:n
2963 }

```

For |

```

2964 \cs_new_protected:Npn \@@_make_m_preamble_iii:n #1
2965 {
2966     \tl_gput_right:Nn \g_@@_preamble_tl { #1 }
2967     \@@_make_m_preamble:n
2968 }

```

For p, m and b

```

2969 \cs_new_protected:Npn \@@_make_m_preamble_iv:nnn #1 #2 #3
2970 {
2971     \tl_gput_right:Nn \g_@@_preamble_tl
2972     {
2973         > {
2974             \@@_cell_begin:w
2975             \begin { minipage } [ #1 ] { \dim_eval:n { #3 } }
2976             \mode_leave_vertical:
2977             \arraybackslash
2978             \vrule height \box_ht:N \@arstrutbox depth 0 pt width 0 pt
2979         }
2980         c
2981         < {
2982             \vrule height 0 pt depth \box_dp:N \@arstrutbox width 0 pt
2983             \end { minipage }
2984             \@@_cell_end:
2985         }
2986     }

```

We test for the presence of a <.

```

2987     \@@_make_m_preamble_x:n
2988 }

```

For w and W

```

2989 \cs_new_protected:Npn \@@_make_m_preamble_v:nnnn #1 #2 #3 #4
2990 {
2991     \tl_gput_right:Nn \g_@@_preamble_tl
2992     {

```

```

2993 > {
2994   \dim_set:Nn \l_@@_col_width_dim { #4 }
2995   \hbox_set:Nw \l_@@_cell_box
2996   \@@_cell_begin:w
2997   \cs_set_nopar:Npn \l_@@_hpos_cell_tl { #3 }
2998 }
2999 c
3000 < {
3001   \@@_cell_end:
3002   \hbox_set_end:
3003   \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
3004   #1
3005   \@@_adjust_size_box:
3006   \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
3007 }
3008 }

```

We test for the presence of a <.

```

3009   \@@_make_m_preamble_x:n
3010 }

```

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

```

3011 \cs_new_protected:Npn \@@_make_m_preamble_x:n #1
3012 {
3013   \str_if_eq:nnTF { #1 } { < }
3014   \@@_make_m_preamble_ix:n
3015   { \@@_make_m_preamble:n { #1 } }
3016 }
3017 \cs_new_protected:Npn \@@_make_m_preamble_ix:n #1
3018 {
3019   \tl_gput_right:Nn \g_@@_preamble_tl { < { #1 } }
3020   \@@_make_m_preamble_x:n
3021 }

```

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).

```

3022 \cs_new_protected:Npn \@@_put_box_in_flow:
3023 {
3024   \box_set_ht:Nn \l_tmpa_box { \box_ht:N \l_tmpa_box + \l_tmpa_dim }
3025   \box_set_dp:Nn \l_tmpa_box { \box_dp:N \l_tmpa_box + \l_tmpb_dim }
3026   \tl_if_eq:NNTF \l_@@_baseline_tl \c_@@_c_tl
3027   { \box_use_drop:N \l_tmpa_box }
3028   \@@_put_box_in_flow_i:
3029 }

```

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

```

3030 \cs_new_protected:Npn \@@_put_box_in_flow_i:
3031 {
3032   \pgfpicture
3033   \@@_qpoint:n { row - 1 }
3034   \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3035   \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
3036   \dim_gadd:Nn \g_tmpa_dim \pgf@y
3037   \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).

```

3038   \tl_if_in:NnTF \l_@@_baseline_tl { line- }
3039   {

```

```

3040     \int_set:Nn \l_tmpa_int
3041     {
3042         \str_range:Nnn
3043         \l_@@_baseline_tl
3044         6
3045         { \tl_count:o \l_@@_baseline_tl }
3046     }
3047     \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
3048 }
3049 {
3050     \tl_if_eq:NnTF \l_@@_baseline_tl { t }
3051     { \int_set_eq:NN \l_tmpa_int \c_one_int }
3052     {
3053         \tl_if_eq:NnTF \l_@@_baseline_tl { b }
3054         { \int_set_eq:NN \l_tmpa_int \c_iRow }
3055         { \int_set:Nn \l_tmpa_int \l_@@_baseline_tl }
3056     }
3057     \bool_lazy_or:nnT
3058     { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
3059     { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
3060     {
3061         \@@_error:n { bad-value-for-baseline }
3062         \int_set_eq:NN \l_tmpa_int \c_one_int
3063     }
3064     \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }

```

We take into account the position of the mathematical axis.

```

3065     \dim_gsub:Nn \g_tmpa_dim { \fontdimen22 \textfont2 }
3066 }
3067 \dim_gsub:Nn \g_tmpa_dim \pgf@y

```

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

```

3068     \endpgfpicture
3069     \box_move_up:nn \g_tmpa_dim { \box_use_drop:N \l_tmpa_box }
3070     \box_use_drop:N \l_tmpa_box
3071 }

```

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).

```

3072 \cs_new_protected:Npn \@@_use_arraybox_with_notes_c:
3073 {

```

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.

```

3074     \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3075     {
3076         \int_compare:nNnT \c_jCol > \c_one_int
3077         {
3078             \box_set_wd:Nn \l_@@_the_array_box
3079             { \box_wd:N \l_@@_the_array_box - \arraycolsep }
3080         }
3081     }

```

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

```

3082     \begin { minipage } [ t ] { \box_wd:N \l_@@_the_array_box }
3083     \bool_if:NT \l_@@_caption_above_bool
3084     {
3085         \tl_if_empty:NF \l_@@_caption_tl
3086         {

```

```

3087         \bool_set_false:N \g_@@_caption_finished_bool
3088         \int_gzero:N \c@tabularnote
3089         \@@_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).

```

3090         \int_compare:nNnT \g_@@_notes_caption_int > \c_zero_int
3091         {
3092             \tl_gput_right:Ne \g_@@_aux_tl
3093             {
3094                 \tl_set:Nn \exp_not:N \l_@@_note_in_caption_tl
3095                 { \int_use:N \g_@@_notes_caption_int }
3096             }
3097             \int_gzero:N \g_@@_notes_caption_int
3098         }
3099     }
3100 }

```

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

```

3101     \hbox
3102     {
3103         \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.

```

3104         \@@_create_extra_nodes:
3105         \seq_if_empty:NF \g_@@_blocks_seq \@@_draw_blocks:
3106     }

```

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).

```

3107     \bool_lazy_any:nT
3108     {
3109         { ! \seq_if_empty_p:N \g_@@_notes_seq }
3110         { ! \seq_if_empty_p:N \g_@@_notes_in_caption_seq }
3111         { ! \tl_if_empty_p:o \g_@@_tabularnote_tl }
3112     }
3113     \@@_insert_tabularnotes:
3114     \cs_set_eq:NN \tabularnote \@@_tabularnote_error:n
3115     \bool_if:NF \l_@@_caption_above_bool \@@_insert_caption:
3116     \end { minipage }
3117 }

```

```

3118 \cs_new_protected:Npn \@@_insert_caption:
3119 {
3120     \tl_if_empty:NF \l_@@_caption_tl
3121     {
3122         \cs_if_exist:NTF \@capttype
3123         { \@@_insert_caption_i: }
3124         { \@@_error:n { caption-outside-float } }
3125     }
3126 }

```

```

3127 \cs_new_protected:Npn \@@_insert_caption_i:
3128 {
3129     \group_begin:

```

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

```
3130 \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.

```
3131 \IfPackageLoadedT { floatrow }
3132 { \cs_set_eq:NN \@makecaption \FR@makecaption }
3133 \tl_if_empty:NTF \l_@@_short_caption_tl
3134 { \caption }
3135 { \caption [ \l_@@_short_caption_tl ] }
3136 { \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.

```
3137 \bool_if:NF \g_@@_caption_finished_bool
3138 {
3139   \bool_gset_true:N \g_@@_caption_finished_bool
3140   \int_gset_eq:NN \g_@@_notes_caption_int \c@tabularnote
3141   \int_gzero:N \c@tabularnote
3142 }
3143 \tl_if_empty:NF \l_@@_label_tl { \label { \l_@@_label_tl } }
3144 \group_end:
3145 }

3146 \cs_new_protected:Npn \@_tabularnote_error:n #1
3147 {
3148   \@_error_or_warning:n { tabularnote~below~the~tabular }
3149   \@_gredirect_none:n { tabularnote~below~the~tabular }
3150 }

3151 \cs_new_protected:Npn \@_insert_tabularnotes:
3152 {
3153   \seq_gconcat:NNN \g_@@_notes_seq \g_@@_notes_in_caption_seq \g_@@_notes_seq
3154   \int_set:Nn \c@tabularnote { \seq_count:N \g_@@_notes_seq }
3155   \skip_vertical:N 0.65ex
```

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

```
3156 \group_begin:
3157 \l_@@_notes_code_before_tl
3158 \tl_if_empty:NF \g_@@_tabularnote_tl
3159 {
3160   \g_@@_tabularnote_tl \par
3161   \tl_gclear:N \g_@@_tabularnote_tl
3162 }
```

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.

```
3163 \int_compare:nNnT \c@tabularnote > \c_zero_int
3164 {
3165   \bool_if:NTF \l_@@_notes_para_bool
3166   {
3167     \begin { tabularnotes* }
3168     \seq_map_inline:Nn \g_@@_notes_seq
3169     { \@_one_tabularnote:nn ##1 }
3170     \strut
3171     \end { tabularnotes* }
3172   }
```

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

```
3172 \par
```



```

3173     }
3174     {
3175         \tabularnotes
3176         \seq_map_inline:Nn \g_@@_notes_seq
3177         { \@@_one_tabularnote:nn ##1 }
3178         \strut
3179         \endtabularnotes
3180     }
3181 }
3182 \unskip
3183 \group_end:
3184 \bool_if:NT \l_@@_notes_bottomrule_bool
3185 {
3186     \IfPackageLoadedTF { booktabs }
3187     {

```

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

```

3188         \skip_vertical:N \aboverulesep

```

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

```

3189         { \CT@arc@ \hrule height \heavyrulewidth }
3190     }
3191     { \@@_error_or_warning:n { bottomrule~without~booktabs } }
3192 }
3193 \l_@@_notes_code_after_tl
3194 \seq_gclear:N \g_@@_notes_seq
3195 \seq_gclear:N \g_@@_notes_in_caption_seq
3196 \int_gzero:N \c@tabularnote
3197 }

```

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.

```

3198 \cs_set_protected:Npn \@@_one_tabularnote:nn #1
3199 {
3200     \tl_if_novalue:nTF { #1 }
3201     { \item }
3202     { \item [ \@@_notes_label_in_list:n { #1 } ] }
3203 }

```

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`.

```

3204 \cs_new_protected:Npn \@@_use_arraybox_with_notes_b:
3205 {
3206     \pgfpicture
3207     \@@_qpoint:n { row - 1 }
3208     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3209     \@@_qpoint:n { row - \int_use:N \c@iRow - base }
3210     \dim_gsub:Nn \g_tmpa_dim \pgf@y
3211     \endpgfpicture
3212     \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3213     \int_if_zero:nT \l_@@_first_row_int
3214     {
3215         \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3216         \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3217     }
3218     \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3219 }

```

Now, the general case.

```

3220 \cs_new_protected:Npn \@@_use_arraybox_with_notes:
3221 {

```

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

```

3222 \tl_if_eq:NnT \l_@@_baseline_tl { t }
3223 { \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`.

```

3224 \pgfpicture
3225 \@@_qpoint:n { row - 1 }
3226 \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3227 \str_if_in:NnTF \l_@@_baseline_tl { line- }
3228 {
3229   \int_set:Nn \l_tmpa_int
3230   {
3231     \str_range:Nnn
3232     \l_@@_baseline_tl
3233     6
3234     { \tl_count:o \l_@@_baseline_tl }
3235   }
3236   \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
3237 }
3238 {
3239   \int_set:Nn \l_tmpa_int \l_@@_baseline_tl
3240   \bool_lazy_or:nnT
3241   { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
3242   { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
3243   {
3244     \@@_error:n { bad-value-for~baseline }
3245     \int_set:Nn \l_tmpa_int 1
3246   }
3247   \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }
3248 }
3249 \dim_gsub:Nn \g_tmpa_dim \pgf@y
3250 \endpgfpicture
3251 \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3252 \int_if_zero:nT \l_@@_first_row_int
3253 {
3254   \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3255   \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3256 }
3257 \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3258 }

```

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.

```

3259 \cs_new_protected:Npn \@@_put_box_in_flow_bis:nn #1 #2
3260 {

```

We will compute the real width of both delimiters used.

```

3261 \dim_zero_new:N \l_@@_real_left_delim_dim
3262 \dim_zero_new:N \l_@@_real_right_delim_dim
3263 \hbox_set:Nn \l_tmpb_box
3264 {
3265   \c_math_toggle_token
3266   \left #1
3267   \vcenter
3268   {
3269     \vbox_to_ht:nn
3270     { \box_ht_plus_dp:N \l_tmpa_box }
3271     { }
3272   }
3273   \right .
3274   \c_math_toggle_token

```

```

3275     }
3276     \dim_set:Nn \l_@@_real_left_delim_dim
3277     { \box_wd:N \l_tmpb_box - \nulldelimiterspace }
3278     \hbox_set:Nn \l_tmpb_box
3279     {
3280       \c_math_toggle_token
3281       \left .
3282       \vbox_to_ht:nn
3283         { \box_ht_plus_dp:N \l_tmpa_box }
3284         { }
3285       \right #2
3286       \c_math_toggle_token
3287     }
3288     \dim_set:Nn \l_@@_real_right_delim_dim
3289     { \box_wd:N \l_tmpb_box - \nulldelimiterspace }

```

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

```

3290     \skip_horizontal:N \l_@@_left_delim_dim
3291     \skip_horizontal:N -\l_@@_real_left_delim_dim
3292     \@@_put_box_in_flow:
3293     \skip_horizontal:N \l_@@_right_delim_dim
3294     \skip_horizontal:N -\l_@@_real_right_delim_dim
3295   }

```

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).

```

3296 \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).

```

3297 {
3298   \peek_remove_spaces:n
3299   {
3300     \peek_meaning:NTF \end
3301     \@@_analyze_end:Nn
3302     {
3303       \@@_transform_preamble:

```

Here is the call to `\array` (we have a dedicated macro `\@@_array`: because of compatibility with the classes `revtex4-1` and `revtex4-2`).

```

3304     \@@_array:o \g_@@_array_preamble_tl
3305   }
3306 }
3307 }
3308 {
3309   \@@_create_col_nodes:
3310   \endarray
3311 }

```

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

```

3312 \NewDocumentEnvironment { @@-light-syntax } { b }
3313 {

```

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.

```

3314   \tl_if_empty:nT { #1 } { \@@_fatal:n { empty-environment } }

```

```

3315 \tl_map_inline:nn { #1 }
3316 {
3317   \str_if_eq:nnT { ##1 } { & }
3318   { \@@_fatal:n { ampersand-in-light-syntax } }
3319   \str_if_eq:nnT { ##1 } { \ }
3320   { \@@_fatal:n { double-backslash-in-light-syntax } }
3321 }

```

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`.

```

3322 \@@_light_syntax_i:w #1 \CodeAfter \q_stop

```

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

```

3323 }

```

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.

```

3324 {
3325   \@@_create_col_nodes:
3326   \endarray
3327 }
3328 \cs_new_protected:Npn \@@_light_syntax_i:w #1\CodeAfter #2\q_stop
3329 {
3330   \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).

```

3331 \seq_clear_new:N \l_@@_rows_seq

```

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

```

3332 \tl_set_rescan:Nno \l_@@_end_of_row_tl { } \l_@@_end_of_row_tl
3333 \bool_if:NTF \l_@@_light_syntax_expanded_bool
3334   \seq_set_split:Nee
3335   \seq_set_split:Non
3336   \l_@@_rows_seq \l_@@_end_of_row_tl { #1 }

```

We delete the last row if it is empty.

```

3337 \seq_pop_right:NN \l_@@_rows_seq \l_tmpa_tl
3338 \tl_if_empty:NF \l_tmpa_tl
3339 { \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.

```

3340 \int_compare:nNnT \l_@@_last_row_int = { -1 }
3341 { \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`).

```

3342 \tl_build_begin:N \l_@@_new_body_tl
3343 \int_zero_new:N \l_@@_nb_cols_int

```

First, we treat the first row.

```

3344 \seq_pop_left:NN \l_@@_rows_seq \l_tmpa_tl
3345 \@@_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).

```

3346 \seq_map_inline:Nn \l_@@_rows_seq
3347 {
3348   \tl_build_put_right:Nn \l_@@_new_body_tl { \\ }
3349   \@@_line_with_light_syntax:n { ##1 }
3350 }
3351 \tl_build_end:N \l_@@_new_body_tl
3352 \int_compare:nNnT \l_@@_last_col_int = { -1 }
3353 {
3354   \int_set:Nn \l_@@_last_col_int
3355     { \l_@@_nb_cols_int - 1 + \l_@@_first_col_int }
3356 }

```

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.

```

3357 \@@_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`).

```

3358 \@@_array:o \g_@@_array_preamble_tl \l_@@_new_body_tl
3359 }
3360 \cs_generate_variant:Nn \@@_line_with_light_syntax:n { o }
3361 \cs_new_protected:Npn \@@_line_with_light_syntax:n #1
3362 {
3363   \seq_clear_new:N \l_@@_cells_seq
3364   \seq_set_split:Nnn \l_@@_cells_seq { ~ } { #1 }
3365   \int_set:Nn \l_@@_nb_cols_int
3366     {
3367       \int_max:nn
3368         \l_@@_nb_cols_int
3369         { \seq_count:N \l_@@_cells_seq }
3370     }
3371   \seq_pop_left:NN \l_@@_cells_seq \l_tmpa_tl
3372   \tl_build_put_right:No \l_@@_new_body_tl \l_tmpa_tl
3373   \seq_map_inline:Nn \l_@@_cells_seq
3374     { \tl_build_put_right:Nn \l_@@_new_body_tl { & ##1 } }
3375 }

```

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`.

```

3376 \cs_new_protected:Npn \@@_analyze_end:Nn #1 #2
3377 {
3378   \str_if_eq:onT \g_@@_name_env_str { #2 }
3379   { \@@_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.

```

3380 \end { #2 }
3381 }

```

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`).

```

3382 \cs_new:Npn \@@_create_col_nodes:
3383 {
3384   \crrc
3385   \int_if_zero:nT \l_@@_first_col_int
3386   {
3387     \omit

```

```

3388 \hbox_overlap_left:n
3389 {
3390   \bool_if:NT \l_@@_code_before_bool
3391   { \pgfsys@markposition { \@@_env: - col - 0 } }
3392   \pgfpicture
3393   \pgfrememberpicturepositiononpagetrue
3394   \pgfcoordinate { \@@_env: - col - 0 } \pgfpointorigin
3395   \str_if_empty:NF \l_@@_name_str
3396   { \pgfnodealias { \l_@@_name_str - col - 0 } { \@@_env: - col - 0 } }
3397   \endpgfpicture
3398   \skip_horizontal:N 2\col@sep
3399   \skip_horizontal:N \g_@@_width_first_col_dim
3400 }
3401 &
3402 }
3403 \omit

```

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

```

3404 \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`).

```

3405 \int_if_zero:nTF \l_@@_first_col_int
3406 {
3407   \bool_if:NT \l_@@_code_before_bool
3408   {
3409     \hbox
3410     {
3411       \skip_horizontal:N -0.5\arrayrulewidth
3412       \pgfsys@markposition { \@@_env: - col - 1 }
3413       \skip_horizontal:N 0.5\arrayrulewidth
3414     }
3415   }
3416   \pgfpicture
3417   \pgfrememberpicturepositiononpagetrue
3418   \pgfcoordinate { \@@_env: - col - 1 }
3419   { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3420   \str_if_empty:NF \l_@@_name_str
3421   { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
3422   \endpgfpicture
3423 }
3424 {
3425   \bool_if:NT \l_@@_code_before_bool
3426   {
3427     \hbox
3428     {
3429       \skip_horizontal:N 0.5\arrayrulewidth
3430       \pgfsys@markposition { \@@_env: - col - 1 }
3431       \skip_horizontal:N -0.5\arrayrulewidth
3432     }
3433   }
3434   \pgfpicture
3435   \pgfrememberpicturepositiononpagetrue
3436   \pgfcoordinate { \@@_env: - col - 1 }
3437   { \pgfpoint { 0.5 \arrayrulewidth } \c_zero_dim }
3438   \str_if_empty:NF \l_@@_name_str
3439   { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
3440   \endpgfpicture
3441 }

```

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.

```

3442 \skip_gset:Nn \g_tmpa_skip { 0 pt~plus 1 fill }
3443 \bool_if:NF \l_@@_auto_columns_width_bool
3444 { \dim_compare:nNnT \l_@@_columns_width_dim > \c_zero_dim }
3445 {
3446   \bool_lazy_and:nnTF
3447     \l_@@_auto_columns_width_bool
3448     { \bool_not_p:n \l_@@_block_auto_columns_width_bool }
3449     { \skip_gadd:Nn \g_tmpa_skip \g_@@_max_cell_width_dim }
3450     { \skip_gadd:Nn \g_tmpa_skip \l_@@_columns_width_dim }
3451     \skip_gadd:Nn \g_tmpa_skip { 2 \col@sep }
3452 }
3453 \skip_horizontal:N \g_tmpa_skip
3454 \hbox
3455 {
3456   \bool_if:NT \l_@@_code_before_bool
3457   {
3458     \hbox
3459     {
3460       \skip_horizontal:N -0.5\arrayrulewidth
3461       \pgfsys@markposition { \@@_env: - col - 2 }
3462       \skip_horizontal:N 0.5\arrayrulewidth
3463     }
3464   }
3465   \pgfpicture
3466   \pgfrememberpicturerepositiononpagetrue
3467   \pgfcoordinate { \@@_env: - col - 2 }
3468   { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3469   \str_if_empty:NF \l_@@_name_str
3470   { \pgfnodealias { \l_@@_name_str - col - 2 } { \@@_env: - col - 2 } }
3471   \endpgfpicture
3472 }

```

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.

```

3473 \int_gset_eq:NN \g_tmpa_int \c_one_int
3474 \bool_if:NTF \g_@@_last_col_found_bool
3475 { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 3 } \c_zero_int } }
3476 { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 2 } \c_zero_int } }
3477 {
3478   &
3479   \omit
3480   \int_gincr:N \g_tmpa_int

```

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

```

3481 \skip_horizontal:N \g_tmpa_skip
3482 \bool_if:NT \l_@@_code_before_bool
3483 {
3484   \hbox
3485   {
3486     \skip_horizontal:N -0.5\arrayrulewidth
3487     \pgfsys@markposition
3488     { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3489     \skip_horizontal:N 0.5\arrayrulewidth
3490   }
3491 }

```

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

```

3492 \pgfpicture
3493 \pgfrememberpicturerepositiononpagetrue
3494 \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3495 { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3496 \str_if_empty:NF \l_@@_name_str

```

```

3497         {
3498             \pgfnodealias
3499             { \l_@@_name_str - col - \int_eval:n { \g_tmpa_int + 1 } }
3500             { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3501         }
3502     \endpgfpicture
3503 }

3504     &
3505     \omit

```

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

```

3506     \int_if_zero:nT \g_@@_col_total_int
3507     { \skip_gset:Nn \g_tmpa_skip { 0 pt+plus 1 fill } }
3508     \skip_horizontal:N \g_tmpa_skip
3509     \int_gincr:N \g_tmpa_int
3510     \bool_lazy_any:nF
3511     {
3512         \g_@@_delims_bool
3513         \l_@@_tabular_bool
3514         { ! \clist_if_empty_p:N \l_@@_vlines_clist }
3515         \l_@@_exterior_arraycolsep_bool
3516         \l_@@_bar_at_end_of_pream_bool
3517     }
3518     { \skip_horizontal:N -\col@sep }
3519     \bool_if:NT \l_@@_code_before_bool
3520     {
3521         \hbox
3522         {
3523             \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.

```

3524         \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3525         { \skip_horizontal:N -\arraycolsep }
3526         \pgfsys@markposition
3527         { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3528         \skip_horizontal:N 0.5\arrayrulewidth
3529         \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3530         { \skip_horizontal:N \arraycolsep }
3531     }
3532 }
3533 \pgfpicture
3534 \pgfrememberpicturepositiononpagetrue
3535 \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3536 {
3537     \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3538     {
3539         \pgfpoint
3540         { - 0.5 \arrayrulewidth - \arraycolsep }
3541         \c_zero_dim
3542     }
3543     { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3544 }
3545 \str_if_empty:NF \l_@@_name_str
3546 {
3547     \pgfnodealias
3548     { \l_@@_name_str - col - \int_eval:n { \g_tmpa_int + 1 } }
3549     { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3550 }
3551 \endpgfpicture

```



```

3552 \bool_if:NT \g_@@_last_col_found_bool
3553 {
3554   \hbox_overlap_right:n
3555   {
3556     \skip_horizontal:N \g_@@_width_last_col_dim
3557     \skip_horizontal:N \col@sep
3558     \bool_if:NT \l_@@_code_before_bool
3559     {
3560       \pgfsys@markposition
3561       { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3562     }
3563     \pgfpicture
3564     \pgfrememberpicturerepositiononpagetrue
3565     \pgfcoordinate
3566     { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3567     \pgfpointorigin
3568     \str_if_empty:NF \l_@@_name_str
3569     {
3570       \pgfnodealias
3571       {
3572         \l_@@_name_str - col
3573         - \int_eval:n { \g_@@_col_total_int + 1 }
3574       }
3575       { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3576     }
3577     \endpgfpicture
3578   }
3579 }
3580 % \cr
3581 }

```

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

```

3582 \tl_const:Nn \c_@@_preamble_first_col_tl
3583 {
3584   >
3585   {

```

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

```

3586 \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
3587 \bool_gset_true:N \g_@@_after_col_zero_bool
3588 \@@_begin_of_row:

```

The contents of the cell is constructed in the box `\l_@@_cell_box` because we have to compute some dimensions of this box.

```

3589 \hbox_set:Nw \l_@@_cell_box
3590 \@@_math_toggle:
3591 \@@_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”.

```

3592 \int_compare:nNnT \c@iRow > \c_zero_int
3593 {
3594   \bool_lazy_or:nnT
3595   { \int_compare_p:nNn \l_@@_last_row_int < \c_zero_int }
3596   { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
3597   {
3598     \l_@@_code_for_first_col_tl
3599     \xglobal \colorlet { nicematrix-first-col } { . }
3600   }
3601 }
3602 }

```

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`.

```

3603     l
3604     <
3605     {
3606         \@@_math_toggle:
3607         \hbox_set_end:
3608         \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
3609         \@@_adjust_size_box:
3610         \@@_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.

```

3611         \dim_gset:Nn \g_@@_width_first_col_dim
3612         { \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.

```

3613         \hbox_overlap_left:n
3614         {
3615             \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > \c_zero_dim
3616             \@@_node_for_cell:
3617             { \box_use_drop:N \l_@@_cell_box }
3618             \skip_horizontal:N \l_@@_left_delim_dim
3619             \skip_horizontal:N \l_@@_left_margin_dim
3620             \skip_horizontal:N \l_@@_extra_left_margin_dim
3621         }
3622         \bool_gset_false:N \g_@@_empty_cell_bool
3623         \skip_horizontal:N -2\col@sep
3624     }
3625 }

```

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

```

3626 \tl_const:Nn \c_@@_preamble_last_col_tl
3627 {
3628     >
3629     {
3630         \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).

```

3631         \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.

```

3632         \bool_gset_true:N \g_@@_last_col_found_bool
3633         \int_gincr:N \c@jCol
3634         \int_gset_eq:NN \g_@@_col_total_int \c@jCol

```

The contents of the cell is constructed in the box `\l_tmpa_box` because we have to compute some dimensions of this box.

```

3635         \hbox_set:Nw \l_@@_cell_box
3636         \@@_math_toggle:
3637         \@@_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”.

```

3638         \int_compare:nNnT \c@iRow > \c_zero_int
3639         {
3640             \bool_lazy_or:nnT
3641             { \int_compare_p:nNn \l_@@_last_row_int < \c_zero_int }
3642             { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
3643             {
3644                 \l_@@_code_for_last_col_tl
3645                 \xglobal \colorlet { nicematrix-last-col } { . }
3646             }
3647         }
3648     }

```

```

3649     1
3650     <
3651     {
3652         \@@_math_toggle:
3653         \hbox_set_end:
3654         \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
3655         \@@_adjust_size_box:
3656         \@@_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.

```

3657         \dim_gset:Nn \g_@@_width_last_col_dim
3658         { \dim_max:nn \g_@@_width_last_col_dim { \box_wd:N \l_@@_cell_box } }
3659         \skip_horizontal:N -2\col@sep

```

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

```

3660         \hbox_overlap_right:n
3661         {
3662             \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \c_zero_dim
3663             {
3664                 \skip_horizontal:N \l_@@_right_delim_dim
3665                 \skip_horizontal:N \l_@@_right_margin_dim
3666                 \skip_horizontal:N \l_@@_extra_right_margin_dim
3667                 \@@_node_for_cell:
3668             }
3669         }
3670         \bool_gset_false:N \g_@@_empty_cell_bool
3671     }
3672 }

```

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

```

3673 \NewDocumentEnvironment { NiceArray } { }
3674 {
3675     \bool_gset_false:N \g_@@_delims_bool
3676     \str_if_empty:NT \g_@@_name_env_str
3677     { \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).

```

3678     \NiceArrayWithDelims . .
3679 }
3680 { \endNiceArrayWithDelims }

```

We create the variants of the environment `{NiceArrayWithDelims}`.

```

3681 \cs_new_protected:Npn \@@_def_env:nnn #1 #2 #3
3682 {
3683     \NewDocumentEnvironment { #1 NiceArray } { }
3684     {
3685         \bool_gset_true:N \g_@@_delims_bool
3686         \str_if_empty:NT \g_@@_name_env_str
3687         { \str_gset:Nn \g_@@_name_env_str { #1 NiceArray } }
3688         \@@_test_if_math_mode:
3689         \NiceArrayWithDelims #2 #3
3690     }
3691     { \endNiceArrayWithDelims }
3692 }
3693 \@@_def_env:nnn p ( )
3694 \@@_def_env:nnn b [ ]
3695 \@@_def_env:nnn B \{ \}
3696 \@@_def_env:nnn v | |
3697 \@@_def_env:nnn V \l \r

```

14 The environment `{NiceMatrix}` and its variants

```

3698 \cs_generate_variant:Nn \@@_begin_of_NiceMatrix:nn { n o }
3699 \cs_new_protected:Npn \@@_begin_of_NiceMatrix:nn #1 #2
3700 {
3701   \bool_set_false:N \l_@@_preamble_bool
3702   \tl_clear:N \l_tmpa_tl
3703   \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3704     { \tl_set:Nn \l_tmpa_tl { @ { } } }
3705   \tl_put_right:Nn \l_tmpa_tl
3706     {
3707       *
3708       {
3709         \int_case:nnF \l_@@_last_col_int
3710         {
3711           { -2 } { \c@MaxMatrixCols }
3712           { -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).

```

3713       }
3714       { \int_eval:n { \l_@@_last_col_int - 1 } }
3715     }
3716     { #2 }
3717   }
3718   \tl_set:Nn \l_tmpb_tl { \use:c { #1 NiceArray } }
3719   \exp_args:No \l_tmpb_tl \l_tmpa_tl
3720 }
3721 \clist_map_inline:nn { p , b , B , v , V }
3722 {
3723   \NewDocumentEnvironment { #1 NiceMatrix } { ! O { } }
3724   {
3725     \bool_gset_true:N \g_@@_delims_bool
3726     \str_gset:Nn \g_@@_name_env_str { #1 NiceMatrix }
3727     \int_if_zero:nT \l_@@_last_col_int
3728     {
3729       \bool_set_true:N \l_@@_last_col_without_value_bool
3730       \int_set:Nn \l_@@_last_col_int { -1 }
3731     }
3732     \keys_set:nn { nicematrix / NiceMatrix } { ##1 }
3733     \@@_begin_of_NiceMatrix:no { #1 } \l_@@_columns_type_tl
3734   }
3735   { \use:c { end #1 NiceArray } }
3736 }

```

We define also an environment `{NiceMatrix}`

```

3737 \NewDocumentEnvironment { NiceMatrix } { ! O { } }
3738 {
3739   \str_gset:Nn \g_@@_name_env_str { NiceMatrix }
3740   \int_if_zero:nT \l_@@_last_col_int
3741   {
3742     \bool_set_true:N \l_@@_last_col_without_value_bool
3743     \int_set:Nn \l_@@_last_col_int { -1 }
3744   }
3745   \keys_set:nn { nicematrix / NiceMatrix } { #1 }
3746   \bool_lazy_or:nnT
3747     { \clist_if_empty_p:N \l_@@_vlines_clist }
3748     { \l_@@_except_borders_bool }
3749     { \bool_set_true:N \l_@@_NiceMatrix_without_vlines_bool }
3750   \@@_begin_of_NiceMatrix:no { } \l_@@_columns_type_tl
3751 }
3752 { \endNiceArray }

```

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

```
3753 \cs_new_protected:Npn \@@_NotEmpty:
3754 { \bool_gset_true:N \g_@@_not_empty_cell_bool }
```

15 {NiceTabular}, {NiceTabularX} and {NiceTabular*}

```
3755 \NewDocumentEnvironment { NiceTabular } { 0 { } m ! 0 { } }
3756 {
```

If the dimension `\l_@@_width_dim` is equal to 0 pt, that means that it has not be set by a previous use of `\NiceMatrixOptions`.

```
3757   \dim_compare:nNnT \l_@@_width_dim = \c_zero_dim
3758     { \dim_set_eq:NN \l_@@_width_dim \linewidth }
3759   \str_gset:Nn \g_@@_name_env_str { NiceTabular }
3760   \keys_set:nn { nicematrix / NiceTabular } { #1 , #3 }
3761   \tl_if_empty:NF \l_@@_short_caption_tl
3762   {
3763     \tl_if_empty:NT \l_@@_caption_tl
3764     {
3765       \@@_error_or_warning:n { short-caption-without-caption }
3766       \tl_set_eq:NN \l_@@_caption_tl \l_@@_short_caption_tl
3767     }
3768   }
3769   \tl_if_empty:NF \l_@@_label_tl
3770   {
3771     \tl_if_empty:NT \l_@@_caption_tl
3772     { \@@_error_or_warning:n { label-without-caption } }
3773   }
3774   \NewDocumentEnvironment { TabularNote } { b }
3775   {
3776     \bool_if:NTF \l_@@_in_code_after_bool
3777       { \@@_error_or_warning:n { TabularNote~in-CodeAfter } }
3778     {
3779       \tl_if_empty:NF \g_@@_tabularnote_tl
3780       { \tl_gput_right:Nn \g_@@_tabularnote_tl { \par } }
3781       \tl_gput_right:Nn \g_@@_tabularnote_tl { ##1 }
3782     }
3783   }
3784   { }
3785   \@@_settings_for_tabular:
3786   \NiceArray { #2 }
3787 }
3788 {
3789   \endNiceArray
3790   \bool_if:NT \c_@@_testphase_table_bool
3791     { \UseTaggingSocket { tbl / hmode / end } }
3792 }
3793 \cs_new_protected:Npn \@@_settings_for_tabular:
3794 {
3795   \bool_set_true:N \l_@@_tabular_bool
3796   \cs_set_eq:NN \@@_math_toggle: \prg_do_nothing:
3797   \cs_set_eq:NN \@@_tuning_not_tabular_begin: \prg_do_nothing:
3798   \cs_set_eq:NN \@@_tuning_not_tabular_end: \prg_do_nothing:
3799 }
```

```
3800 \NewDocumentEnvironment { NiceTabularX } { m 0 { } m ! 0 { } }
3801 {
3802   \str_gset:Nn \g_@@_name_env_str { NiceTabularX }
3803   \dim_zero_new:N \l_@@_width_dim
3804   \dim_set:Nn \l_@@_width_dim { #1 }
3805   \keys_set:nn { nicematrix / NiceTabular } { #2 , #4 }
3806   \@@_settings_for_tabular:
```

```

3807 \NiceArray { #3 }
3808 }
3809 {
3810 \endNiceArray
3811 \int_if_zero:nT \g_@@_total_X_weight_int
3812 { \@@_error:n { NiceTabularX~without~X } }
3813 }

3814 \NewDocumentEnvironment { NiceTabular* } { m O { } m ! O { } }
3815 {
3816 \str_gset:Nn \g_@@_name_env_str { NiceTabular* }
3817 \dim_set:Nn \l_@@_tabular_width_dim { #1 }
3818 \keys_set:nn { nicematrix / NiceTabular } { #2 , #4 }
3819 \@@_settings_for_tabular:
3820 \NiceArray { #3 }
3821 }
3822 { \endNiceArray }

```

16 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`).

```

3823 \cs_new_protected:Npn \@@_deal_with_rounded_corners:
3824 {
3825 \bool_lazy_all:nT
3826 {
3827 { \int_compare_p:nNn \l_@@_tab_rounded_corners_dim > \c_zero_dim }
3828 \l_@@_hvlines_bool
3829 { ! \g_@@_delims_bool }
3830 { ! \l_@@_except_borders_bool }
3831 }
3832 {
3833 \bool_set_true:N \l_@@_except_borders_bool
3834 \clist_if_empty:NF \l_@@_corners_clist
3835 { \@@_error:n { hvlines,~rounded-corners-and~corners } }
3836 \tl_gput_right:Nn \g_@@_pre_code_after_tl
3837 {
3838 \@@_stroke_block:nnn
3839 {
3840 rounded-corners = \dim_use:N \l_@@_tab_rounded_corners_dim ,
3841 draw = \l_@@_rules_color_tl
3842 }
3843 { 1-1 }
3844 { \int_use:N \c@iRow - \int_use:N \c@jCol }
3845 }
3846 }
3847 }

3848 \cs_new_protected:Npn \@@_after_array:
3849 {

```

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`.

```

3850 \hook_gremove_code:nn { env / tabular / begin } { nicematrix }
3851 \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.

```

3852     \bool_if:NT \g_@@_last_col_found_bool
3853     { \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`.

```

3854     \bool_if:NT \l_@@_last_col_without_value_bool
3855     { \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.

```

3856     \bool_if:NT \l_@@_last_row_without_value_bool
3857     { \int_set_eq:NN \l_@@_last_row_int \g_@@_row_total_int }

```

```

3858     \tl_gput_right:Ne \g_@@_aux_tl
3859     {
3860       \seq_gset_from_clist:Nn \exp_not:N \g_@@_size_seq
3861       {
3862         \int_use:N \l_@@_first_row_int ,
3863         \int_use:N \c@iRow ,
3864         \int_use:N \g_@@_row_total_int ,
3865         \int_use:N \l_@@_first_col_int ,
3866         \int_use:N \c@jCol ,
3867         \int_use:N \g_@@_col_total_int
3868       }
3869     }

```

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`).

```

3870     \seq_if_empty:NF \g_@@_pos_of_blocks_seq
3871     {
3872       \tl_gput_right:Ne \g_@@_aux_tl
3873       {
3874         \seq_gset_from_clist:Nn \exp_not:N \g_@@_pos_of_blocks_seq
3875         { \seq_use:Nnnn \g_@@_pos_of_blocks_seq , , , }
3876       }
3877     }
3878     \seq_if_empty:NF \g_@@_multicolumn_cells_seq
3879     {
3880       \tl_gput_right:Ne \g_@@_aux_tl
3881       {
3882         \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_cells_seq
3883         { \seq_use:Nnnn \g_@@_multicolumn_cells_seq , , , }
3884         \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_sizes_seq
3885         { \seq_use:Nnnn \g_@@_multicolumn_sizes_seq , , , }
3886       }
3887     }

```

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

```

3888     \@@_create_diag_nodes:

```

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

```

3889     \pgfpicture
3890     \int_step_inline:nn \c@iRow
3891     {
3892       \pgfnodealias
3893       { \@@_env: - ##1 - last }
3894       { \@@_env: - ##1 - \int_use:N \c@jCol }

```

```

3895     }
3896     \int_step_inline:nn \c@jCol
3897     {
3898         \pgfnodealias
3899         { \l_@@_env: - last - ##1 }
3900         { \l_@@_env: - \int_use:N \c@iRow - ##1 }
3901     }
3902     \str_if_empty:NF \l_@@_name_str
3903     {
3904         \int_step_inline:nn \c@iRow
3905         {
3906             \pgfnodealias
3907             { \l_@@_name_str - ##1 - last }
3908             { \l_@@_env: - ##1 - \int_use:N \c@jCol }
3909         }
3910         \int_step_inline:nn \c@jCol
3911         {
3912             \pgfnodealias
3913             { \l_@@_name_str - last - ##1 }
3914             { \l_@@_env: - \int_use:N \c@iRow - ##1 }
3915         }
3916     }
3917     \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.

```

3918     \bool_if:NT \l_@@_parallelize_diags_bool
3919     {
3920         \int_gzero_new:N \g_@@_ddots_int
3921         \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.

```

3922         \dim_gzero_new:N \g_@@_delta_x_one_dim
3923         \dim_gzero_new:N \g_@@_delta_y_one_dim
3924         \dim_gzero_new:N \g_@@_delta_x_two_dim
3925         \dim_gzero_new:N \g_@@_delta_y_two_dim
3926     }
3927     \int_zero_new:N \l_@@_initial_i_int
3928     \int_zero_new:N \l_@@_initial_j_int
3929     \int_zero_new:N \l_@@_final_i_int
3930     \int_zero_new:N \l_@@_final_j_int
3931     \bool_set_false:N \l_@@_initial_open_bool
3932     \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.

```

3933     \bool_if:NT \l_@@_small_bool
3934     {
3935         \dim_set:Nn \l_@@_xdots_radius_dim { 0.7 \l_@@_xdots_radius_dim }
3936         \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.

```

3937         \dim_set:Nn \l_@@_xdots_shorten_start_dim
3938         { 0.6 \l_@@_xdots_shorten_start_dim }
3939         \dim_set:Nn \l_@@_xdots_shorten_end_dim

```

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


```

3940         { 0.6 \l_@@_xdots_shorten_end_dim }
3941     }

```

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

```

3942     \@@_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_seq` which will contain all the cells which are empty (and not in a block) considered in the corners of the array.

```

3943     \@@_compute_corners:

```

The sequence `\g_@@_pos_of_blocks_seq` must be “adjusted” (for the case where the user have written something like `\Block{1-*}`).

```

3944     \@@_adjust_pos_of_blocks_seq:
3945     \@@_deal_with_rounded_corners:
3946     \tl_if_empty:NF \l_@@_hlines_clist \@@_draw_hlines:
3947     \tl_if_empty:NF \l_@@_vlines_clist \@@_draw_vlines:

```

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

```

3948     \IfPackageLoadedT { tikz }
3949     {
3950         \tikzset
3951         {
3952             every-picture / .style =
3953             {
3954                 overlay ,
3955                 remember~picture ,
3956                 name~prefix = \@@_env: -
3957             }
3958         }
3959     }
3960     \bool_if:NT \c_@@_tagging_array_bool
3961     { \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign: }
3962     \cs_set_eq:NN \SubMatrix \@@_SubMatrix
3963     \cs_set_eq:NN \UnderBrace \@@_UnderBrace
3964     \cs_set_eq:NN \OverBrace \@@_OverBrace
3965     \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
3966     \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
3967     \cs_set_eq:NN \line \@@_line
3968     \g_@@_pre_code_after_tl
3969     \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.

```

3970     \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).

```

3971     \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`).

```

3972     % \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
3973     % { \@@_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:`.

```

3974 \bool_set_true:N \l_@@_in_code_after_bool
3975 \exp_last_unbraced:No \@@_CodeAfter_keys: \g_nicematrix_code_after_tl
3976 \scan_stop:
3977 \tl_gclear:N \g_nicematrix_code_after_tl
3978 \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.

```

3979 \seq_if_empty:NF \g_@@_rowlistcolors_seq { \@@_clear_rowlistcolors_seq: }
3980 \tl_if_empty:NF \g_@@_pre_code_before_tl
3981 {
3982   \tl_gput_right:Ne \g_@@_aux_tl
3983   {
3984     \tl_gset:Nn \exp_not:N \g_@@_pre_code_before_tl
3985     { \exp_not:o \g_@@_pre_code_before_tl }
3986   }
3987   \tl_gclear:N \g_@@_pre_code_before_tl
3988 }
3989 \tl_if_empty:NF \g_nicematrix_code_before_tl
3990 {
3991   \tl_gput_right:Ne \g_@@_aux_tl
3992   {
3993     \tl_gset:Nn \exp_not:N \g_@@_code_before_tl
3994     { \exp_not:o \g_nicematrix_code_before_tl }
3995   }
3996   \tl_gclear:N \g_nicematrix_code_before_tl
3997 }

3998 \str_gclear:N \g_@@_name_env_str
3999 \@@_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`.

```

4000 \cs_gset_eq:NN \CT@arc@ \@@_old_CT@arc@
4001 }

```

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`.

```

4002 \NewDocumentCommand \@@_CodeAfter_keys: { 0 { } }
4003 { \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).

```

4004 \cs_new_protected:Npn \@@_adjust_pos_of_blocks_seq:
4005 {

```

¹²e.g. `\color[rgb]{0.5,0.5,0}`

```

4006 \seq_gset_map_x:NNn \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq
4007 { \@@_adjust_pos_of_blocks_seq_i:nnnnn ##1 }
4008 }

```

The following command must *not* be protected.

```

4009 \cs_new:Npn \@@_adjust_pos_of_blocks_seq_i:nnnnn #1 #2 #3 #4 #5
4010 {
4011   { #1 }
4012   { #2 }
4013   {
4014     \int_compare:nNnTF { #3 } > { 99 }
4015     { \int_use:N \c@iRow }
4016     { #3 }
4017   }
4018   {
4019     \int_compare:nNnTF { #4 } > { 99 }
4020     { \int_use:N \c@jCol }
4021     { #4 }
4022   }
4023   { #5 }
4024 }

```

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).

```

4025 \hook_gput_code:nnn { begindocument } { . }
4026 {
4027   \cs_new_protected:Npx \@@_draw_dotted_lines:
4028   {
4029     \c_@@_pgfortikzpicture_tl
4030     \@@_draw_dotted_lines_i:
4031     \c_@@_endpgfortikzpicture_tl
4032   }
4033 }

```

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

```

4034 \cs_new_protected:Npn \@@_draw_dotted_lines_i:
4035 {
4036   \pgfrememberpicturerepositiononpagetrue
4037   \pgf@relevantforpicturesizefalse
4038   \g_@@_HVdotsfor_lines_tl
4039   \g_@@_Vdots_lines_tl
4040   \g_@@_Ddots_lines_tl
4041   \g_@@_Iddots_lines_tl
4042   \g_@@_Cdots_lines_tl
4043   \g_@@_Ldots_lines_tl
4044 }

4045 \cs_new_protected:Npn \@@_restore_iRow_jCol:
4046 {
4047   \cs_if_exist:NT \theiRow { \int_gset_eq:NN \c@iRow \l_@@_old_iRow_int }
4048   \cs_if_exist:NT \thejCol { \int_gset_eq:NN \c@jCol \l_@@_old_jCol_int }
4049 }

```

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

```

4050 \pgfdeclareshape { @@_diag_node }
4051 {
4052   \savedanchor { \five }
4053   {
4054     \dim_gset_eq:NN \pgf@x \l_tmpa_dim

```

```

4055     \dim_gset_eq:NN \pgf@y \l_tmpb_dim
4056   }
4057   \anchor { 5 } { \five }
4058   \anchor { center } { \pgfpointorigin }
4059   \anchor { 1 } { \five \pgf@x = 0.2 \pgf@x \pgf@y = 0.2 \pgf@y }
4060   \anchor { 2 } { \five \pgf@x = 0.4 \pgf@x \pgf@y = 0.4 \pgf@y }
4061   \anchor { 3 } { \five \pgf@x = 0.6 \pgf@x \pgf@y = 0.6 \pgf@y }
4062   \anchor { 4 } { \five \pgf@x = 0.8 \pgf@x \pgf@y = 0.8 \pgf@y }
4063   \anchor { 6 } { \five \pgf@x = 1.2 \pgf@x \pgf@y = 1.2 \pgf@y }
4064   \anchor { 7 } { \five \pgf@x = 1.4 \pgf@x \pgf@y = 1.4 \pgf@y }
4065   \anchor { 8 } { \five \pgf@x = 1.6 \pgf@x \pgf@y = 1.6 \pgf@y }
4066   \anchor { 9 } { \five \pgf@x = 1.8 \pgf@x \pgf@y = 1.8 \pgf@y }
4067 }

```

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).

```

4068 \cs_new_protected:Npn \@@_create_diag_nodes:
4069 {
4070   \pgfpicture
4071   \pgfrememberpicturerepositiononpagetrue
4072   \int_step_inline:nn { \int_max:nn \c@iRow \c@jCol }
4073   {
4074     \@@_qpoint:n { col - \int_min:nn { ##1 } { \c@jCol + 1 } }
4075     \dim_set_eq:NN \l_tmpa_dim \pgf@x
4076     \@@_qpoint:n { row - \int_min:nn { ##1 } { \c@iRow + 1 } }
4077     \dim_set_eq:NN \l_tmpb_dim \pgf@y
4078     \@@_qpoint:n { col - \int_min:nn { ##1 + 1 } { \c@jCol + 1 } }
4079     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
4080     \@@_qpoint:n { row - \int_min:nn { ##1 + 1 } { \c@iRow + 1 } }
4081     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
4082     \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.

```

4083     \dim_set:Nn \l_tmpa_dim { ( \l_@@_tmpc_dim - \l_tmpa_dim ) / 2 }
4084     \dim_set:Nn \l_tmpb_dim { ( \l_@@_tmpd_dim - \l_tmpb_dim ) / 2 }
4085     \pgfnode { @@_diag_node } { center } { } { \@@_env: - ##1 } { }
4086     \str_if_empty:NF \l_@@_name_str
4087     { \pgfnodealias { \l_@@_name_str - ##1 } { \@@_env: - ##1 } }
4088   }

```

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

```

4089   \int_set:Nn \l_tmpa_int { \int_max:nn \c@iRow \c@jCol + 1 }
4090   \@@_qpoint:n { row - \int_min:nn { \l_tmpa_int } { \c@iRow + 1 } }
4091   \dim_set_eq:NN \l_tmpa_dim \pgf@y
4092   \@@_qpoint:n { col - \int_min:nn { \l_tmpa_int } { \c@jCol + 1 } }
4093   \pgfcoordinate
4094   { \@@_env: - \int_use:N \l_tmpa_int } { \pgfpoint \pgf@x \l_tmpa_dim }
4095   \pgfnodealias
4096   { \@@_env: - last }
4097   { \@@_env: - \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
4098   \str_if_empty:NF \l_@@_name_str
4099   {
4100     \pgfnodealias
4101     { \l_@@_name_str - \int_use:N \l_tmpa_int }
4102     { \@@_env: - \int_use:N \l_tmpa_int }
4103     \pgfnodealias
4104     { \l_@@_name_str - last }
4105     { \@@_env: - last }
4106   }
4107   \endpgfpicture
4108 }

```

17 We draw the dotted lines

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

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

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

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

This command computes:

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

```
4109 \cs_new_protected:Npn \@@_find_extremities_of_line:nnnn #1 #2 #3 #4
4110 {
```

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

```
4111 \cs_set:cpn { @@ _ dotted _ #1 - #2 } { }
```

Initialization of variables.

```
4112 \int_set:Nn \l_@@_initial_i_int { #1 }
4113 \int_set:Nn \l_@@_initial_j_int { #2 }
4114 \int_set:Nn \l_@@_final_i_int { #1 }
4115 \int_set:Nn \l_@@_final_j_int { #2 }
```

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

```
4116 \bool_set_false:N \l_@@_stop_loop_bool
4117 \bool_do_until:Nn \l_@@_stop_loop_bool
4118 {
4119   \int_add:Nn \l_@@_final_i_int { #3 }
4120   \int_add:Nn \l_@@_final_j_int { #4 }
4121   \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.

```
4122 \if_int_compare:w \l_@@_final_i_int > \l_@@_row_max_int
4123   \if_int_compare:w #3 = \c_one_int
4124     \bool_set_true:N \l_@@_final_open_bool
4125   \else:
4126     \if_int_compare:w \l_@@_final_j_int > \l_@@_col_max_int
4127       \bool_set_true:N \l_@@_final_open_bool
4128     \fi:
4129   \fi:
4130 \else:
4131   \if_int_compare:w \l_@@_final_j_int < \l_@@_col_min_int
```

```

4132         \if_int_compare:w #4 = -1
4133         \bool_set_true:N \l_@@_final_open_bool
4134         \fi:
4135     \else:
4136         \if_int_compare:w \l_@@_final_j_int > \l_@@_col_max_int
4137         \if_int_compare:w #4 = \c_one_int
4138         \bool_set_true:N \l_@@_final_open_bool
4139         \fi:
4140     \fi:
4141 \fi:
4142 \fi:
4143 \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.

```

4144     {

```

We do a step backwards.

```

4145         \int_sub:Nn \l_@@_final_i_int { #3 }
4146         \int_sub:Nn \l_@@_final_j_int { #4 }
4147         \bool_set_true:N \l_@@_stop_loop_bool
4148     }

```

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`.

```

4149     {
4150         \cs_if_exist:cTF
4151         {
4152             @@ _ dotted _
4153             \int_use:N \l_@@_final_i_int -
4154             \int_use:N \l_@@_final_j_int
4155         }
4156         {
4157             \int_sub:Nn \l_@@_final_i_int { #3 }
4158             \int_sub:Nn \l_@@_final_j_int { #4 }
4159             \bool_set_true:N \l_@@_final_open_bool
4160             \bool_set_true:N \l_@@_stop_loop_bool
4161         }
4162         {
4163             \cs_if_exist:cTF
4164             {
4165                 pgf @ sh @ ns @ \@@_env:
4166                 - \int_use:N \l_@@_final_i_int
4167                 - \int_use:N \l_@@_final_j_int
4168             }
4169             { \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.

```

4170         {
4171             \cs_set:cpn
4172             {
4173                 @@ _ dotted _
4174                 \int_use:N \l_@@_final_i_int -
4175                 \int_use:N \l_@@_final_j_int
4176             }
4177             { }
4178         }
4179     }
4180 }
4181 }

```

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

```
4182 \bool_set_false:N \l_@@_stop_loop_bool
```

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

```
4183 \int_set:Nn \l_tmpa_int { \l_@@_col_min_int - 1 }
4184 \bool_do_until:Nn \l_@@_stop_loop_bool
4185 {
4186   \int_sub:Nn \l_@@_initial_i_int { #3 }
4187   \int_sub:Nn \l_@@_initial_j_int { #4 }
4188   \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.

```
4189 \if_int_compare:w \l_@@_initial_i_int < \l_@@_row_min_int
4190 \if_int_compare:w #3 = \c_one_int
4191 \bool_set_true:N \l_@@_initial_open_bool
4192 \else:
```

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

```
4193 \if_int_compare:w \l_@@_initial_j_int = \l_tmpa_int
4194 \bool_set_true:N \l_@@_initial_open_bool
4195 \fi:
4196 \fi:
4197 \else:
4198 \if_int_compare:w \l_@@_initial_j_int < \l_@@_col_min_int
4199 \if_int_compare:w #4 = \c_one_int
4200 \bool_set_true:N \l_@@_initial_open_bool
4201 \fi:
4202 \else:
4203 \if_int_compare:w \l_@@_initial_j_int > \l_@@_col_max_int
4204 \if_int_compare:w #4 = -1
4205 \bool_set_true:N \l_@@_initial_open_bool
4206 \fi:
4207 \fi:
4208 \fi:
4209 \fi:
4210 \bool_if:NTF \l_@@_initial_open_bool
4211 {
4212   \int_add:Nn \l_@@_initial_i_int { #3 }
4213   \int_add:Nn \l_@@_initial_j_int { #4 }
4214   \bool_set_true:N \l_@@_stop_loop_bool
4215 }
4216 {
4217   \cs_if_exist:cTF
4218   {
4219     @@ _ dotted _
4220     \int_use:N \l_@@_initial_i_int -
4221     \int_use:N \l_@@_initial_j_int
4222   }
4223   {
4224     \int_add:Nn \l_@@_initial_i_int { #3 }
4225     \int_add:Nn \l_@@_initial_j_int { #4 }
4226     \bool_set_true:N \l_@@_initial_open_bool
4227     \bool_set_true:N \l_@@_stop_loop_bool
4228   }
4229   {
4230     \cs_if_exist:cTF
4231     {
4232       pgf @ sh @ ns @ \@@_env:
4233       - \int_use:N \l_@@_initial_i_int
4234       - \int_use:N \l_@@_initial_j_int
4235     }
```

```

4236         { \bool_set_true:N \l_@@_stop_loop_bool }
4237         {
4238             \cs_set:cpn
4239             {
4240                 @@ _ dotted _
4241                 \int_use:N \l_@@_initial_i_int -
4242                 \int_use:N \l_@@_initial_j_int
4243             }
4244             { }
4245         }
4246     }
4247 }
4248 }

```

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.

```

4249 \seq_gput_right:Ne \g_@@_pos_of_xdots_seq
4250 {
4251     { \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`.

```

4252     { \int_min:nn \l_@@_initial_j_int \l_@@_final_j_int }
4253     { \int_use:N \l_@@_final_i_int }
4254     { \int_max:nn \l_@@_initial_j_int \l_@@_final_j_int }
4255     { } % for the name of the block
4256 }
4257 }

```

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.

```

4258 \cs_new_protected:Npn \@@_open_shorten:
4259 {
4260     \bool_if:NT \l_@@_initial_open_bool
4261     { \dim_zero:N \l_@@_xdots_shorten_start_dim }
4262     \bool_if:NT \l_@@_final_open_bool
4263     { \dim_zero:N \l_@@_xdots_shorten_end_dim }
4264 }

```

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).

```

4265 \cs_new_protected:Npn \@@_adjust_to_submatrix:nn #1 #2
4266 {
4267     \int_set:Nn \l_@@_row_min_int 1
4268     \int_set:Nn \l_@@_col_min_int 1
4269     \int_set_eq:NN \l_@@_row_max_int \c@iRow
4270     \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`.

```

4271 \seq_map_inline:Nn \g_@@_submatrix_seq
4272 { \@@_adjust_to_submatrix:nnnnnn { #1 } { #2 } ##1 }
4273 }

```

#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.


```

4274 \cs_set_protected:Npn \@@_adjust_to_submatrix:nnnnnn #1 #2 #3 #4 #5 #6
4275 {
4276   \int_compare:nNf { #3 } > { #1 }
4277   {
4278     \int_compare:nNf { #1 } > { #5 }
4279     {
4280       \int_compare:nNf { #4 } > { #2 }
4281       {
4282         \int_compare:nNf { #2 } > { #6 }
4283         {
4284           \int_set:Nn \l_@@_row_min_int
4285             { \int_max:nn \l_@@_row_min_int { #3 } }
4286           \int_set:Nn \l_@@_col_min_int
4287             { \int_max:nn \l_@@_col_min_int { #4 } }
4288           \int_set:Nn \l_@@_row_max_int
4289             { \int_min:nn \l_@@_row_max_int { #5 } }
4290           \int_set:Nn \l_@@_col_max_int
4291             { \int_min:nn \l_@@_col_max_int { #6 } }
4292         }
4293       }
4294     }
4295   }
4296 }

4297 \cs_new_protected:Npn \@@_set_initial_coords:
4298 {
4299   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4300   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4301 }
4302 \cs_new_protected:Npn \@@_set_final_coords:
4303 {
4304   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4305   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4306 }
4307 \cs_new_protected:Npn \@@_set_initial_coords_from_anchor:n #1
4308 {
4309   \pgfpointanchor
4310   {
4311     \@@_env:
4312     - \int_use:N \l_@@_initial_i_int
4313     - \int_use:N \l_@@_initial_j_int
4314   }
4315   { #1 }
4316   \@@_set_initial_coords:
4317 }
4318 \cs_new_protected:Npn \@@_set_final_coords_from_anchor:n #1
4319 {
4320   \pgfpointanchor
4321   {
4322     \@@_env:
4323     - \int_use:N \l_@@_final_i_int
4324     - \int_use:N \l_@@_final_j_int
4325   }
4326   { #1 }
4327   \@@_set_final_coords:
4328 }

4329 \cs_new_protected:Npn \@@_open_x_initial_dim:
4330 {
4331   \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
4332   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
4333   {
4334     \cs_if_exist:cT
4335     { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }

```

```

4336     {
4337         \pgfpointanchor
4338         { \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4339         { west }
4340         \dim_set:Nn \l_@@_x_initial_dim
4341         { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
4342     }
4343 }

```

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

```

4344 \dim_compare:nNtT \l_@@_x_initial_dim = \c_max_dim
4345 {
4346     \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4347     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4348     \dim_add:Nn \l_@@_x_initial_dim \col@sep
4349 }
4350 }
4351 \cs_new_protected:Npn \@@_open_x_final_dim:
4352 {
4353     \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
4354     \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
4355     {
4356         \cs_if_exist:cT
4357         { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4358         {
4359             \pgfpointanchor
4360             { \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4361             { east }
4362             \dim_set:Nn \l_@@_x_final_dim
4363             { \dim_max:nn \l_@@_x_final_dim \pgf@x }
4364         }
4365     }

```

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

```

4366 \dim_compare:nNtT \l_@@_x_final_dim = { - \c_max_dim }
4367 {
4368     \@@_qpoint:n { col - \int_eval:n { \l_@@_final_j_int + 1 } }
4369     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4370     \dim_sub:Nn \l_@@_x_final_dim \col@sep
4371 }
4372 }

```

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.

```

4373 \cs_new_protected:Npn \@@_draw_Ldots:nnn #1 #2 #3
4374 {
4375     \@@_adjust_to_submatrix:nn { #1 } { #2 }
4376     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4377     {
4378         \@@_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.

```

4379 \group_begin:
4380 \@@_open_shorten:
4381 \int_if_zero:nTF { #1 }
4382 { \color { nicematrix-first-row } }
4383 {

```

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.

```

4384 \int_compare:nNtT { #1 } = \l_@@_last_row_int
4385 { \color { nicematrix-last-row } }

```

```

4386         }
4387         \keys_set:nn { nicematrix / xdots } { #3 }
4388         \tl_if_empty:oF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4389         \@@_actually_draw_Ldots:
4390     \group_end:
4391 }
4392 }

```

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`.

```

4393 \cs_new_protected:Npn \@@_actually_draw_Ldots:
4394 {
4395     \bool_if:NTF \l_@@_initial_open_bool
4396     {
4397         \@@_open_x_initial_dim:
4398         \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4399         \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4400     }
4401     { \@@_set_initial_coords_from_anchor:n { base-east } }
4402     \bool_if:NTF \l_@@_final_open_bool
4403     {
4404         \@@_open_x_final_dim:
4405         \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4406         \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4407     }
4408     { \@@_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.

```

4409     \bool_lazy_all:nTF
4410     {
4411         \l_@@_initial_open_bool
4412         \l_@@_final_open_bool
4413         { \int_compare_p:nNn \l_@@_initial_i_int = \l_@@_last_row_int }
4414     }
4415     {
4416         \dim_add:Nn \l_@@_y_initial_dim \c_@@_shift_Ldots_last_row_dim
4417         \dim_add:Nn \l_@@_y_final_dim \c_@@_shift_Ldots_last_row_dim
4418     }

```

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 (?).

```

4419     {
4420         \dim_add:Nn \l_@@_y_initial_dim \l_@@_xdots_radius_dim
4421         \dim_add:Nn \l_@@_y_final_dim \l_@@_xdots_radius_dim
4422     }
4423     \@@_draw_line:
4424 }

```

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.

```

4425 \cs_new_protected:Npn \@@_draw_Cdots:nnn #1 #2 #3
4426 {
4427   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4428   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4429   {
4430     \@@_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.

```

4431   \group_begin:
4432   \@@_open_shorten:
4433   \int_if_zero:nTF { #1 }
4434   { \color { nicematrix-first-row } }
4435   {

```

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.

```

4436       \int_compare:nNnT { #1 } = \l_@@_last_row_int
4437       { \color { nicematrix-last-row } }
4438   }
4439   \keys_set:nn { nicematrix / xdots } { #3 }
4440   \tl_if_empty:oF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4441   \@@_actually_draw_Cdots:
4442   \group_end:
4443 }
4444 }

```

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`.

```

4445 \cs_new_protected:Npn \@@_actually_draw_Cdots:
4446 {
4447   \bool_if:NTF \l_@@_initial_open_bool
4448   { \@@_open_x_initial_dim: }
4449   { \@@_set_initial_coords_from_anchor:n { mid-east } }
4450   \bool_if:NTF \l_@@_final_open_bool
4451   { \@@_open_x_final_dim: }
4452   { \@@_set_final_coords_from_anchor:n { mid-west } }
4453   \bool_lazy_and:nnTF
4454   \l_@@_initial_open_bool
4455   \l_@@_final_open_bool
4456   {
4457     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int }
4458     \dim_set_eq:NN \l_tmpa_dim \pgf@y
4459     \@@_qpoint:n { row - \int_eval:n { \l_@@_initial_i_int + 1 } }
4460     \dim_set:Nn \l_@@_y_initial_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
4461     \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
4462   }
4463   {
4464     \bool_if:NT \l_@@_initial_open_bool
4465     { \dim_set_eq:NN \l_@@_y_initial_dim \l_@@_y_final_dim }
4466     \bool_if:NT \l_@@_final_open_bool
4467     { \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim }

```

```

4468     }
4469     \@@_draw_line:
4470 }
4471 \cs_new_protected:Npn \@@_open_y_initial_dim:
4472 {
4473     \dim_set:Nn \l_@@_y_initial_dim { - \c_max_dim }
4474     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
4475     {
4476         \cs_if_exist:cT
4477         { pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4478         {
4479             \pgfpointanchor
4480             { \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4481             { north }
4482             \dim_set:Nn \l_@@_y_initial_dim
4483             { \dim_max:nn \l_@@_y_initial_dim \pgf@y }
4484         }
4485     }
4486     \dim_compare:nNnT \l_@@_y_initial_dim = { - \c_max_dim }
4487     {
4488         \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4489         \dim_set:Nn \l_@@_y_initial_dim
4490         {
4491             \fp_to_dim:n
4492             {
4493                 \pgf@y
4494                 + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
4495             }
4496         }
4497     }
4498 }
4499 \cs_new_protected:Npn \@@_open_y_final_dim:
4500 {
4501     \dim_set_eq:NN \l_@@_y_final_dim \c_max_dim
4502     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
4503     {
4504         \cs_if_exist:cT
4505         { pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4506         {
4507             \pgfpointanchor
4508             { \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4509             { south }
4510             \dim_set:Nn \l_@@_y_final_dim
4511             { \dim_min:nn \l_@@_y_final_dim \pgf@y }
4512         }
4513     }
4514     \dim_compare:nNnT \l_@@_y_final_dim = \c_max_dim
4515     {
4516         \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4517         \dim_set:Nn \l_@@_y_final_dim
4518         { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
4519     }
4520 }

```

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.

```

4521 \cs_new_protected:Npn \@@_draw_Vdots:nnn #1 #2 #3
4522 {
4523     \@@_adjust_to_submatrix:nn { #1 } { #2 }
4524     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4525     {
4526         \@@_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.

```

4527 \group_begin:
4528 \@@_open_shorten:
4529 \int_if_zero:nTF { #2 }
4530 { \color { nicematrix-first-col } }
4531 {
4532 \int_compare:nNtT { #2 } = \l_@@_last_col_int
4533 { \color { nicematrix-last-col } }
4534 }
4535 \keys_set:nn { nicematrix / xdots } { #3 }
4536 \tl_if_empty:oF \l_@@_xdots_color_tl
4537 { \color { \l_@@_xdots_color_tl } }
4538 \@@_actually_draw_Vdots:
4539 \group_end:
4540 }
4541 }

```

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`.

```

4542 \cs_new_protected:Npn \@@_actually_draw_Vdots:
4543 {

```

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

```

4544 \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.

```

4545 {
4546 \@@_open_y_initial_dim:
4547 \@@_open_y_final_dim:
4548 \int_if_zero:nTF \l_@@_initial_j_int

```

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

```

4549 {
4550 \@@_qpoint:n { col - 1 }
4551 \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4552 \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
4553 \dim_sub:Nn \l_@@_x_initial_dim \l_@@_extra_left_margin_dim
4554 \dim_sub:Nn \l_@@_x_initial_dim \c_@@_shift_exterior_Vdots_dim
4555 }
4556 {
4557 \bool_lazy_and:nnTF
4558 { \int_compare_p:nNn \l_@@_last_col_int > { -2 } }
4559 { \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”.

```

4560 {
4561 \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4562 \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4563 \dim_add:Nn \l_@@_x_initial_dim \l_@@_right_margin_dim
4564 \dim_add:Nn \l_@@_x_initial_dim \l_@@_extra_right_margin_dim
4565 \dim_add:Nn \l_@@_x_initial_dim \c_@@_shift_exterior_Vdots_dim
4566 }

```

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

```

4567         {
4568             \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4569             \dim_set_eq:NN \l_tmpa_dim \pgf@x
4570             \@@_qpoint:n { col - \int_eval:n { \l_@@_initial_j_int + 1 } }
4571             \dim_set:Nn \l_@@_x_initial_dim { ( \pgf@x + \l_tmpa_dim ) / 2 }
4572         }
4573     }
4574 }

```

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.

```

4575     {
4576         \bool_set_false:N \l_tmpa_bool
4577         \bool_if:NF \l_@@_initial_open_bool
4578         {
4579             \bool_if:NF \l_@@_final_open_bool
4580             {
4581                 \@@_set_initial_coords_from_anchor:n { south-west }
4582                 \@@_set_final_coords_from_anchor:n { north-west }
4583                 \bool_set:Nn \l_tmpa_bool
4584                     { \dim_compare_p:nNn \l_@@_x_initial_dim = \l_@@_x_final_dim }
4585             }
4586         }

```

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

```

4587         \bool_if:NTF \l_@@_initial_open_bool
4588         {
4589             \@@_open_y_initial_dim:
4590             \@@_set_final_coords_from_anchor:n { north }
4591             \dim_set_eq:NN \l_@@_x_initial_dim \l_@@_x_final_dim
4592         }
4593         {
4594             \@@_set_initial_coords_from_anchor:n { south }
4595             \bool_if:NTF \l_@@_final_open_bool
4596             \@@_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.

```

4597         {
4598             \@@_set_final_coords_from_anchor:n { north }
4599             \dim_compare:nNnF \l_@@_x_initial_dim = \l_@@_x_final_dim
4600             {
4601                 \dim_set:Nn \l_@@_x_initial_dim
4602                 {
4603                     \bool_if:NTF \l_tmpa_bool \dim_min:nn \dim_max:nn
4604                     \l_@@_x_initial_dim \l_@@_x_final_dim
4605                 }
4606             }
4607         }
4608     }
4609 }
4610 \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
4611 \@@_draw_line:
4612 }

```

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.

```

4613 \cs_new_protected:Npn \@@_draw_Ddots:nnn #1 #2 #3
4614 {

```

```

4615 \@@_adjust_to_submatrix:nn { #1 } { #2 }
4616 \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4617 {
4618 \@@_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.

```

4619 \group_begin:
4620 \@@_open_shorten:
4621 \keys_set:nn { nicematrix / xdots } { #3 }
4622 \tl_if_empty:oF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4623 \@@_actually_draw_Ddots:
4624 \group_end:
4625 }
4626 }

```

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.`

```

4627 \cs_new_protected:Npn \@@_actually_draw_Ddots:
4628 {
4629 \bool_if:NTF \l_@@_initial_open_bool
4630 {
4631 \@@_open_y_initial_dim:
4632 \@@_open_x_initial_dim:
4633 }
4634 { \@@_set_initial_coords_from_anchor:n { south-east } }
4635 \bool_if:NTF \l_@@_final_open_bool
4636 {
4637 \@@_open_x_final_dim:
4638 \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4639 }
4640 { \@@_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.

```

4641 \bool_if:NT \l_@@_parallelize_diags_bool
4642 {
4643 \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).

```

4644 \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.

```

4645 {
4646 \dim_gset:Nn \g_@@_delta_x_one_dim
4647 { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4648 \dim_gset:Nn \g_@@_delta_y_one_dim
4649 { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4650 }

```


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`.

```

4651     {
4652         \dim_compare:nNnF \g_@@_delta_x_one_dim = \c_zero_dim
4653         {
4654             \dim_set:Nn \l_@@_y_final_dim
4655             {
4656                 \l_@@_y_initial_dim +
4657                 ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4658                 \dim_ratio:nn \g_@@_delta_y_one_dim \g_@@_delta_x_one_dim
4659             }
4660         }
4661     }
4662 }
4663 \@@_draw_line:
4664 }

```

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.

```

4665 \cs_new_protected:Npn \@@_draw_Iddots:nnn #1 #2 #3
4666 {
4667     \@@_adjust_to_submatrix:nn { #1 } { #2 }
4668     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4669     {
4670         \@@_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.

```

4671     \group_begin:
4672     \@@_open_shorten:
4673     \keys_set:nn { nicematrix / xdots } { #3 }
4674     \tl_if_empty:oF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4675     \@@_actually_draw_Iddots:
4676     \group_end:
4677 }
4678 }

```

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`.

```

4679 \cs_new_protected:Npn \@@_actually_draw_Iddots:
4680 {
4681     \bool_if:NTF \l_@@_initial_open_bool
4682     {
4683         \@@_open_y_initial_dim:
4684         \@@_open_x_initial_dim:
4685     }
4686     { \@@_set_initial_coords_from_anchor:n { south~west } }
4687     \bool_if:NTF \l_@@_final_open_bool
4688     {
4689         \@@_open_y_final_dim:
4690         \@@_open_x_final_dim:

```

```

4691     }
4692     { \@@_set_final_coords_from_anchor:n { north-east } }
4693     \bool_if:NT \l_@@_parallelize_diags_bool
4694     {
4695         \int_gincr:N \g_@@_iddots_int
4696         \int_compare:nNnTF \g_@@_iddots_int = \c_one_int
4697         {
4698             \dim_gset:Nn \g_@@_delta_x_two_dim
4699             { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4700             \dim_gset:Nn \g_@@_delta_y_two_dim
4701             { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4702         }
4703         {
4704             \dim_compare:nNnF \g_@@_delta_x_two_dim = \c_zero_dim
4705             {
4706                 \dim_set:Nn \l_@@_y_final_dim
4707                 {
4708                     \l_@@_y_initial_dim +
4709                     ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4710                     \dim_ratio:nn \g_@@_delta_y_two_dim \g_@@_delta_x_two_dim
4711                 }
4712             }
4713         }
4714     }
4715     \@@_draw_line:
4716 }

```

18 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`

```

4717 \cs_new_protected:Npn \@@_draw_line:
4718 {
4719     \pgfrememberpicturepositiononpagetrue
4720     \pgf@relevantforpicturesizefalse
4721     \bool_lazy_or:nnTF
4722     { \tl_if_eq_p:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl }
4723     \l_@@_dotted_bool
4724     \@@_draw_standard_dotted_line:
4725     \@@_draw_unstandard_dotted_line:
4726 }

```

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.

```

4727 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:
4728 {

```

```

4729 \begin { scope }
4730 \@@_draw_unstandard_dotted_line:o
4731 { \l_@@_xdots_line_style_tl , \l_@@_xdots_color_tl }
4732 }

```

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.

```

4733 \cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:n { o }
4734 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:n #1
4735 {
4736   \@@_draw_unstandard_dotted_line:nooo
4737   { #1 }
4738   \l_@@_xdots_up_tl
4739   \l_@@_xdots_down_tl
4740   \l_@@_xdots_middle_tl
4741 }

```

The following Tikz styles are for the three labels (set by the symbols `_`, `^` and `=`) of a continuous line with a non-standard style.

```

4742 \hook_gput_code:nnn { begindocument } { . }
4743 {
4744   \IfPackageLoadedT { tikz }
4745   {
4746     \tikzset
4747     {
4748       @@_node_above / .style = { sloped , above } ,
4749       @@_node_below / .style = { sloped , below } ,
4750       @@_node_middle / .style =
4751       {
4752         sloped ,
4753         inner~sep = \c_@@_innersep_middle_dim
4754       }
4755     }
4756   }
4757 }

4758 \cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:nnnn { n o o o }
4759 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:nnnn #1 #2 #3 #4
4760 {

```

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.

```

4761 \dim_zero_new:N \l_@@_l_dim
4762 \dim_set:Nn \l_@@_l_dim
4763 {
4764   \fp_to_dim:n
4765   {
4766     sqrt
4767     (
4768       ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
4769       +
4770       ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
4771     )
4772   }
4773 }

```

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.

```

4774 \dim_compare:nNnT \l_@@_l_dim < \c_@@_max_l_dim
4775 {
4776   \dim_compare:nNnT \l_@@_l_dim > { 1 pt }
4777   \@@_draw_unstandard_dotted_line_i:
4778 }

```

If the key `xdots/horizontal-labels` has been used.

```

4779 \bool_if:NT \l_@@_xdots_h_labels_bool
4780 {
4781   \tikzset
4782   {
4783     @@_node_above / .style = { auto = left } ,
4784     @@_node_below / .style = { auto = right } ,
4785     @@_node_middle / .style = { innersep = \c_@@_innersep_middle_dim }
4786   }
4787 }
4788 \tl_if_empty:nF { #4 }
4789 { \tikzset { @@_node_middle / .append~style = { fill = white } } }
4790 \draw
4791 [ #1 ]
4792 ( \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).

```

4793 -- node [ @@_node_middle ] { $ \scriptstyle #4 $ }
4794 node [ @@_node_below ] { $ \scriptstyle #3 $ }
4795 node [ @@_node_above ] { $ \scriptstyle #2 $ }
4796 ( \l_@@_x_final_dim , \l_@@_y_final_dim ) ;
4797 \end { scope }
4798 }
4799 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line_i:
4800 {
4801   \dim_set:Nn \l_tmpa_dim
4802   {
4803     \l_@@_x_initial_dim
4804     + ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
4805     * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
4806   }
4807   \dim_set:Nn \l_tmpb_dim
4808   {
4809     \l_@@_y_initial_dim
4810     + ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
4811     * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
4812   }
4813   \dim_set:Nn \l_@@_tmpc_dim
4814   {
4815     \l_@@_x_final_dim
4816     - ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
4817     * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
4818   }
4819   \dim_set:Nn \l_@@_tmpd_dim
4820   {
4821     \l_@@_y_final_dim
4822     - ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
4823     * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
4824   }
4825   \dim_set_eq:NN \l_@@_x_initial_dim \l_tmpa_dim
4826   \dim_set_eq:NN \l_@@_y_initial_dim \l_tmpb_dim
4827   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_tmpc_dim

```

```

4828 \dim_set_eq:NN \l_@@_y_final_dim \l_@@_tmpd_dim
4829 }

```

The command `\@@_draw_standard_dotted_line:` draws the line with our system of dots (which gives a dotted line with real rounded dots).

```

4830 \cs_new_protected:Npn \@@_draw_standard_dotted_line:
4831 {
4832   \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.

```

4833   \dim_zero_new:N \l_@@_l_dim
4834   \dim_set:Nn \l_@@_l_dim
4835   {
4836     \fp_to_dim:n
4837     {
4838       sqrt
4839       (
4840         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
4841         +
4842         ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
4843       )
4844     }
4845   }

```

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.

```

4846   \dim_compare:nNnT \l_@@_l_dim < \c_@@_max_l_dim
4847   {
4848     \dim_compare:nNnT \l_@@_l_dim > { 1 pt }
4849     \@@_draw_standard_dotted_line_i:
4850   }
4851   \group_end:
4852   \bool_lazy_all:nF
4853   {
4854     { \tl_if_empty_p:N \l_@@_xdots_up_tl }
4855     { \tl_if_empty_p:N \l_@@_xdots_down_tl }
4856     { \tl_if_empty_p:N \l_@@_xdots_middle_tl }
4857   }
4858   \l_@@_labels_standard_dotted_line:
4859 }
4860 \dim_const:Nn \c_@@_max_l_dim { 50 cm }
4861 \cs_new_protected:Npn \@@_draw_standard_dotted_line_i:
4862 {

```

The number of dots will be `\l_tmpa_int + 1`.

```

4863   \int_set:Nn \l_tmpa_int
4864   {
4865     \dim_ratio:nn
4866     {
4867       \l_@@_l_dim
4868       - \l_@@_xdots_shorten_start_dim
4869       - \l_@@_xdots_shorten_end_dim
4870     }
4871     \l_@@_xdots_inter_dim
4872   }

```

The dimensions `\l_tmpa_dim` and `\l_tmpb_dim` are the coordinates of the vector between two dots in the dotted line.

```

4873   \dim_set:Nn \l_tmpa_dim

```

```

4874 {
4875   ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4876   \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
4877 }
4878 \dim_set:Nn \l_tmpb_dim
4879 {
4880   ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
4881   \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
4882 }

```

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.

```

4883 \dim_gadd:Nn \l_@@_x_initial_dim
4884 {
4885   ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4886   \dim_ratio:nn
4887   {
4888     \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
4889     + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
4890   }
4891   { 2 \l_@@_l_dim }
4892 }
4893 \dim_gadd:Nn \l_@@_y_initial_dim
4894 {
4895   ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
4896   \dim_ratio:nn
4897   {
4898     \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
4899     + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
4900   }
4901   { 2 \l_@@_l_dim }
4902 }
4903 \pgf@relevantforpicturesizefalse
4904 \int_step_inline:nnn \c_zero_int \l_tmpa_int
4905 {
4906   \pgfpathcircle
4907   { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
4908   { \l_@@_xdots_radius_dim }
4909   \dim_add:Nn \l_@@_x_initial_dim \l_tmpa_dim
4910   \dim_add:Nn \l_@@_y_initial_dim \l_tmpb_dim
4911 }
4912 \pgfusepathqfill
4913 }

```

```

4914 \cs_new_protected:Npn \l_@@_labels_standard_dotted_line:
4915 {
4916   \pgfscope
4917   \pgftransformshift
4918   {
4919     \pgfpointlineattime { 0.5 }
4920     { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
4921     { \pgfpoint \l_@@_x_final_dim \l_@@_y_final_dim }
4922   }
4923   \fp_set:Nn \l_tmpa_fp
4924   {
4925     atand
4926     (
4927       \l_@@_y_final_dim - \l_@@_y_initial_dim ,
4928       \l_@@_x_final_dim - \l_@@_x_initial_dim
4929     )
4930   }
4931   \pgftransformrotate { \fp_use:N \l_tmpa_fp }
4932   \bool_if:NF \l_@@_xdots_h_labels_bool { \fp_zero:N \l_tmpa_fp }

```

```

4933 \tl_if_empty:NF \l_@@_xdots_middle_tl
4934 {
4935   \begin { pgfscope }
4936   \pgfset { inner~sep = \c_@@_innersep_middle_dim }
4937   \pgfnode
4938   { rectangle }
4939   { center }
4940   {
4941     \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4942     {
4943       \c_math_toggle_token
4944       \scriptstyle \l_@@_xdots_middle_tl
4945       \c_math_toggle_token
4946     }
4947   }
4948   { }
4949   {
4950     \pgfsetfillcolor { white }
4951     \pgfusepath { fill }
4952   }
4953   \end { pgfscope }
4954 }
4955 \tl_if_empty:NF \l_@@_xdots_up_tl
4956 {
4957   \pgfnode
4958   { rectangle }
4959   { south }
4960   {
4961     \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4962     {
4963       \c_math_toggle_token
4964       \scriptstyle \l_@@_xdots_up_tl
4965       \c_math_toggle_token
4966     }
4967   }
4968   { }
4969   { \pgfusepath { } }
4970 }
4971 \tl_if_empty:NF \l_@@_xdots_down_tl
4972 {
4973   \pgfnode
4974   { rectangle }
4975   { north }
4976   {
4977     \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4978     {
4979       \c_math_toggle_token
4980       \scriptstyle \l_@@_xdots_down_tl
4981       \c_math_toggle_token
4982     }
4983   }
4984   { }
4985   { \pgfusepath { } }
4986 }
4987 \endpgfscope
4988 }

```

19 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.

```

4989 \hook_gput_code:nnn { begindocument } { . }
4990 {
4991   \cs_set_nopar:Npn \l_@@_argspec_tl { m E { _ ^ : } { { } { } { } } }
4992   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
4993   \cs_new_protected:Npn \@@_Ldots
4994     { \@@_collect_options:n { \@@_Ldots_i } }
4995   \exp_args:NNo \NewDocumentCommand \@@_Ldots_i \l_@@_argspec_tl
4996     {
4997       \int_if_zero:nTF \c@jCol
4998       { \@@_error:nn { in~first~col } \Ldots }
4999       {
5000         \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
5001         { \@@_error:nn { in~last~col } \Ldots }
5002         {
5003           \@@_instruction_of_type:nnn \c_false_bool { Ldots }
5004           { #1 , down = #2 , up = #3 , middle = #4 }
5005         }
5006       }
5007       \bool_if:NF \l_@@_nullify_dots_bool
5008       { \phantom { \ensuremath { \@@_old_ldots } } }
5009       \bool_gset_true:N \g_@@_empty_cell_bool
5010     }

5011   \cs_new_protected:Npn \@@_Cdots
5012     { \@@_collect_options:n { \@@_Cdots_i } }
5013   \exp_args:NNo \NewDocumentCommand \@@_Cdots_i \l_@@_argspec_tl
5014     {
5015       \int_if_zero:nTF \c@jCol
5016       { \@@_error:nn { in~first~col } \Cdots }
5017       {
5018         \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
5019         { \@@_error:nn { in~last~col } \Cdots }
5020         {
5021           \@@_instruction_of_type:nnn \c_false_bool { Cdots }
5022           { #1 , down = #2 , up = #3 , middle = #4 }
5023         }
5024       }
5025       \bool_if:NF \l_@@_nullify_dots_bool
5026       { \phantom { \ensuremath { \@@_old_cdots } } }
5027       \bool_gset_true:N \g_@@_empty_cell_bool
5028     }

5029   \cs_new_protected:Npn \@@_Vdots
5030     { \@@_collect_options:n { \@@_Vdots_i } }
5031   \exp_args:NNo \NewDocumentCommand \@@_Vdots_i \l_@@_argspec_tl
5032     {
5033       \int_if_zero:nTF \c@iRow
5034       { \@@_error:nn { in~first~row } \Vdots }
5035       {

```



```

5036 \int_compare:nNnTF \c@iRow = \l_@@_last_row_int
5037 { \@@_error:nn { in~last~row } \Vdots }
5038 {
5039 \@@_instruction_of_type:nnn \c_false_bool { Vdots }
5040 { #1 , down = #2 , up = #3 , middle = #4 }
5041 }
5042 }
5043 \bool_if:NF \l_@@_nullify_dots_bool
5044 { \phantom { \ensuremath { \@@_old_vdots } } }
5045 \bool_gset_true:N \g_@@_empty_cell_bool
5046 }

5047 \cs_new_protected:Npn \@@_Ddots
5048 { \@@_collect_options:n { \@@_Ddots_i } }
5049 \exp_args:NNo \NewDocumentCommand \@@_Ddots_i \l_@@_argspec_tl
5050 {
5051 \int_case:nnF \c@iRow
5052 {
5053 0 { \@@_error:nn { in~first~row } \Ddots }
5054 \l_@@_last_row_int { \@@_error:nn { in~last~row } \Ddots }
5055 }
5056 {
5057 \int_case:nnF \c@jCol
5058 {
5059 0 { \@@_error:nn { in~first~col } \Ddots }
5060 \l_@@_last_col_int { \@@_error:nn { in~last~col } \Ddots }
5061 }
5062 {
5063 \keys_set_known:nn { nicematrix / Ddots } { #1 }
5064 \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Ddots }
5065 { #1 , down = #2 , up = #3 , middle = #4 }
5066 }
5067 }
5068 }
5069 \bool_if:NF \l_@@_nullify_dots_bool
5070 { \phantom { \ensuremath { \@@_old_ddots } } }
5071 \bool_gset_true:N \g_@@_empty_cell_bool
5072 }

5073 \cs_new_protected:Npn \@@_Iddots
5074 { \@@_collect_options:n { \@@_Iddots_i } }
5075 \exp_args:NNo \NewDocumentCommand \@@_Iddots_i \l_@@_argspec_tl
5076 {
5077 \int_case:nnF \c@iRow
5078 {
5079 0 { \@@_error:nn { in~first~row } \Iddots }
5080 \l_@@_last_row_int { \@@_error:nn { in~last~row } \Iddots }
5081 }
5082 {
5083 \int_case:nnF \c@jCol
5084 {
5085 0 { \@@_error:nn { in~first~col } \Iddots }
5086 \l_@@_last_col_int { \@@_error:nn { in~last~col } \Iddots }
5087 }
5088 {
5089 \keys_set_known:nn { nicematrix / Ddots } { #1 }
5090 \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Iddots }
5091 { #1 , down = #2 , up = #3 , middle = #4 }
5092 }
5093 }
5094 \bool_if:NF \l_@@_nullify_dots_bool
5095 { \phantom { \ensuremath { \@@_old_iddots } } }

```

```

5096     \bool_gset_true:N \g_@@_empty_cell_bool
5097   }
5098 }

```

End of the \AddToHook.

Despite its name, the following set of keys will be used for \Ddots but also for \Iddots.

```

5099 \keys_define:nn { nicematrix / Ddots }
5100 {
5101   draw-first .bool_set:N = \l_@@_draw_first_bool ,
5102   draw-first .default:n = true ,
5103   draw-first .value_forbidden:n = true
5104 }

```

The command \@@_Hspace: will be linked to \hspace in {NiceArray}.

```

5105 \cs_new_protected:Npn \@@_Hspace:
5106 {
5107   \bool_gset_true:N \g_@@_empty_cell_bool
5108   \hspace
5109 }

```

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.

```

5110 \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.

```

5111 \cs_new:Npn \@@_Hdotsfor:
5112 {
5113   \bool_lazy_and:nnTF
5114     { \int_if_zero_p:n \c@jCol }
5115     { \int_if_zero_p:n \l_@@_first_col_int }
5116   {
5117     \bool_if:NTF \g_@@_after_col_zero_bool
5118     {
5119       \multicolumn { 1 } { c } { }
5120       \@@_Hdotsfor_i
5121     }
5122     { \@@_fatal:n { Hdotsfor~in~col~0 } }
5123   }
5124   {
5125     \multicolumn { 1 } { c } { }
5126     \@@_Hdotsfor_i
5127   }
5128 }

```

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:).

```

5129 \hook_gput_code:nnn { begindocument } { . }
5130 {
5131   \cs_set_nopar:Npn \l_@@_argspec_tl { m m 0 { } E { _ ^ : } { { } { } { } } }
5132   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl

```

We don't put ! before the last optionnal argument for homogeneity with \Cdots, etc. which have only one optional argument.

```

5133   \cs_new_protected:Npn \@@_Hdotsfor_i
5134     { \@@_collect_options:n { \@@_Hdotsfor_ii } }
5135   \exp_args:NNo \NewDocumentCommand \@@_Hdotsfor_ii \l_@@_argspec_tl
5136     {

```

```

5137 \tl_gput_right:Ne \g_@@_HVDotsfor_lines_tl
5138 {
5139   \@@_Hdotsfor:nnnn
5140   { \int_use:N \c@iRow }
5141   { \int_use:N \c@jCol }
5142   { #2 }
5143   {
5144     #1 , #3 ,
5145     down = \exp_not:n { #4 } ,
5146     up = \exp_not:n { #5 } ,
5147     middle = \exp_not:n { #6 }
5148   }
5149 }
5150 \prg_replicate:nn { #2 - 1 }
5151 {
5152   &
5153   \multicolumn { 1 } { c } { }
5154   \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
5155 }
5156 }
5157 }

```

```

5158 \cs_new_protected:Npn \@@_Hdotsfor:nnnn #1 #2 #3 #4
5159 {
5160   \bool_set_false:N \l_@@_initial_open_bool
5161   \bool_set_false:N \l_@@_final_open_bool

```

For the row, it's easy.

```

5162 \int_set:Nn \l_@@_initial_i_int { #1 }
5163 \int_set_eq:NN \l_@@_final_i_int \l_@@_initial_i_int

```

For the column, it's a bit more complicated.

```

5164 \int_compare:nNnTF { #2 } = \c_one_int
5165 {
5166   \int_set_eq:NN \l_@@_initial_j_int \c_one_int
5167   \bool_set_true:N \l_@@_initial_open_bool
5168 }
5169 {
5170   \cs_if_exist:cTF
5171   {
5172     pgf @ sh @ ns @ \@@_env:
5173     - \int_use:N \l_@@_initial_i_int
5174     - \int_eval:n { #2 - 1 }
5175   }
5176   { \int_set:Nn \l_@@_initial_j_int { #2 - 1 } }
5177   {
5178     \int_set:Nn \l_@@_initial_j_int { #2 }
5179     \bool_set_true:N \l_@@_initial_open_bool
5180   }
5181 }
5182 \int_compare:nNnTF { #2 + #3 - 1 } = \c_jCol
5183 {
5184   \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5185   \bool_set_true:N \l_@@_final_open_bool
5186 }
5187 {
5188   \cs_if_exist:cTF
5189   {
5190     pgf @ sh @ ns @ \@@_env:
5191     - \int_use:N \l_@@_final_i_int
5192     - \int_eval:n { #2 + #3 }
5193   }
5194   { \int_set:Nn \l_@@_final_j_int { #2 + #3 } }
5195   {

```

```

5196         \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5197         \bool_set_true:N \l_@@_final_open_bool
5198     }
5199 }

5200 \group_begin:
5201 \@@_open_shorten:
5202 \int_if_zero:nTF { #1 }
5203 { \color { nicematrix-first-row } }
5204 {
5205     \int_compare:nNnT { #1 } = \g_@@_row_total_int
5206     { \color { nicematrix-last-row } }
5207 }

5208
5209 \keys_set:nn { nicematrix / xdots } { #4 }
5210 \tl_if_empty:oF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
5211 \@@_actually_draw_Ldots:
5212 \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).

```

5213     \int_step_inline:nnn { #2 } { #2 + #3 - 1 }
5214     { \cs_set:cpn { @@ _ dotted _ #1 - ##1 } { } }
5215 }

5216 \hook_gput_code:nnn { begindocument } { . }
5217 {
5218     \cs_set_nopar:Npn \l_@@_argspec_tl { m m 0 { } E { _ ^ : } { { } { } { } } }
5219     \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
5220     \cs_new_protected:Npn \@@_Vdotsfor:
5221     { \@@_collect_options:n { \@@_Vdotsfor_i } }
5222     \exp_args:NNo \NewDocumentCommand \@@_Vdotsfor_i \l_@@_argspec_tl
5223     {
5224         \bool_gset_true:N \g_@@_empty_cell_bool
5225         \tl_gput_right:Ne \g_@@_HVdotsfor_lines_tl
5226         {
5227             \@@_Vdotsfor:nnnn
5228             { \int_use:N \c@iRow }
5229             { \int_use:N \c@jCol }
5230             { #2 }
5231             {
5232                 #1 , #3 ,
5233                 down = \exp_not:n { #4 } ,
5234                 up = \exp_not:n { #5 } ,
5235                 middle = \exp_not:n { #6 }
5236             }
5237         }
5238     }
5239 }

5240 \cs_new_protected:Npn \@@_Vdotsfor:nnnn #1 #2 #3 #4
5241 {
5242     \bool_set_false:N \l_@@_initial_open_bool
5243     \bool_set_false:N \l_@@_final_open_bool

```

For the column, it’s easy.

```

5244     \int_set:Nn \l_@@_initial_j_int { #2 }
5245     \int_set_eq:NN \l_@@_final_j_int \l_@@_initial_j_int

```

For the row, it's a bit more complicated.

```

5246 \int_compare:nNnTF { #1 } = \c_one_int
5247 {
5248   \int_set_eq:NN \l_@@_initial_i_int \c_one_int
5249   \bool_set_true:N \l_@@_initial_open_bool
5250 }
5251 {
5252   \cs_if_exist:cTF
5253   {
5254     pgf @ sh @ ns @ \@@_env:
5255     - \int_eval:n { #1 - 1 }
5256     - \int_use:N \l_@@_initial_j_int
5257   }
5258   { \int_set:Nn \l_@@_initial_i_int { #1 - 1 } }
5259   {
5260     \int_set:Nn \l_@@_initial_i_int { #1 }
5261     \bool_set_true:N \l_@@_initial_open_bool
5262   }
5263 }
5264 \int_compare:nNnTF { #1 + #3 - 1 } = \c_iRow
5265 {
5266   \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5267   \bool_set_true:N \l_@@_final_open_bool
5268 }
5269 {
5270   \cs_if_exist:cTF
5271   {
5272     pgf @ sh @ ns @ \@@_env:
5273     - \int_eval:n { #1 + #3 }
5274     - \int_use:N \l_@@_final_j_int
5275   }
5276   { \int_set:Nn \l_@@_final_i_int { #1 + #3 } }
5277   {
5278     \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5279     \bool_set_true:N \l_@@_final_open_bool
5280   }
5281 }
5282 \group_begin:
5283 \@@_open_shorten:
5284 \int_if_zero:nTF { #2 }
5285 { \color { nicematrix-first-col } }
5286 {
5287   \int_compare:nNnT { #2 } = \g_@@_col_total_int
5288   { \color { nicematrix-last-col } }
5289 }
5290 \keys_set:nn { nicematrix / xdots } { #4 }
5291 \tl_if_empty:oF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
5292 \@@_actually_draw_Vdots:
5293 \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).

```

5294 \int_step_inline:nnn { #1 } { #1 + #3 - 1 }
5295 { \cs_set:cpn { @@ _ dotted _ ##1 - #2 } { } }
5296 }

```

The command `\@@_rotate:` will be linked to `\rotate` in `{NiceArrayWithDelims}`.

```

5297 \NewDocumentCommand \@@_rotate: { 0 { } }
5298 {

```

```

5299 \peek_remove_spaces:n
5300 {
5301     \bool_gset_true:N \g_@@_rotate_bool
5302     \keys_set:nn { nicematrix / rotate } { #1 }
5303 }
5304 }

5305 \keys_define:nn { nicematrix / rotate }
5306 {
5307     c .code:n = \bool_gset_true:N \g_@@_rotate_c_bool ,
5308     c .value_forbidden:n = true ,
5309     unknown .code:n = \@@_error:n { Unknown~key~for~rotate }
5310 }

```

20 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

```

5311 \cs_new:Npn \@@_double_int_eval:n #1-#2 \q_stop
5312 {
5313     \tl_if_empty:nTF { #2 }
5314     { #1 }
5315     { \@@_double_int_eval_i:n #1-#2 \q_stop }
5316 }
5317 \cs_new:Npn \@@_double_int_eval_i:n #1-#2- \q_stop
5318 { \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).

```

5319 \hook_gput_code:nnn { begindocument } { . }
5320 {
5321     \cs_set_nopar:Npn \l_@@_argspec_tl
5322     { 0 { } m m ! 0 { } E { _ ^ : } { { } { } { } } }
5323     \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
5324     \exp_args:NNo \NewDocumentCommand \@@_line \l_@@_argspec_tl
5325     {
5326         \group_begin:
5327         \keys_set:nn { nicematrix / xdots } { #1 , #4 , down = #5 , up = #6 }
5328         \tl_if_empty:oF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
5329         \use:e
5330         {
5331             \@@_line_i:nn
5332             { \@@_double_int_eval:n #2 - \q_stop }
5333             { \@@_double_int_eval:n #3 - \q_stop }

```

¹³Indeed, we want that the user may use the command `\line` in `\CodeAfter` with LaTeX counters in the arguments — with the command `\value`.

```

5334     }
5335     \group_end:
5336 }
5337 }

5338 \cs_new_protected:Npn \@@_line_i:nn #1 #2
5339 {
5340     \bool_set_false:N \l_@@_initial_open_bool
5341     \bool_set_false:N \l_@@_final_open_bool
5342     \bool_lazy_or:nnTF
5343     { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #1 } }
5344     { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #2 } }
5345     { \@@_error:nnn { unknown-cell-for-line-in-CodeAfter } { #1 } { #2 } }

```

The test of `measuring@` is a security (cf. question 686649 on TeX StackExchange).

```

5346     { \legacy_if:nF { measuring@ } { \@@_draw_line_ii:nn { #1 } { #2 } } }
5347 }

5348 \hook_gput_code:nnn { begindocument } { . }
5349 {
5350     \cs_new_protected:Npx \@@_draw_line_ii:nn #1 #2
5351     {

```

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:`.

```

5352         \c_@@_pgfortikzpicture_tl
5353         \@@_draw_line_iii:nn { #1 } { #2 }
5354         \c_@@_endpgfortikzpicture_tl
5355     }
5356 }

```

The following command *must* be protected (it’s used in the construction of `\@@_draw_line_ii:nn`).

```

5357 \cs_new_protected:Npn \@@_draw_line_iii:nn #1 #2
5358 {
5359     \pgfrememberpicturepositiononpagetrue
5360     \pgfpointshapeborder { \@@_env: - #1 } { \@@_qpoint:n { #2 } }
5361     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
5362     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
5363     \pgfpointshapeborder { \@@_env: - #2 } { \@@_qpoint:n { #1 } }
5364     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
5365     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
5366     \@@_draw_line:
5367 }

```

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).

21 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

```
5368 \cs_new:Npn \@@_if_row_less_than:nn #1 #2
5369 { \int_compare:nNtT { \c@iRow } < { #1 } { #2 } }
```

\@@_put_in_row_style will be used several times by \RowStyle.

```
5370 \cs_generate_variant:Nn \@@_put_in_row_style:n { e }
5371 \cs_set_protected:Npn \@@_put_in_row_style:n #1
5372 {
5373   \tl_gput_right:Ne \g_@@_row_style_tl
5374   {
```

Be careful, \exp_not:N \@@_if_row_less_than:nn can't be replaced by a protected version of \@@_if_row_less_than:nn.

```
5375     \exp_not:N
5376     \@@_if_row_less_than:nn
5377     { \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).

```
5378     { \exp_not:n { #1 } \scan_stop: }
5379   }
5380 }
```

```
5381 \keys_define:nn { nicematrix / RowStyle }
5382 {
5383   cell-space-top-limit .dim_set:N = \l_tmpa_dim ,
5384   cell-space-top-limit .value_required:n = true ,
5385   cell-space-bottom-limit .dim_set:N = \l_tmpb_dim ,
5386   cell-space-bottom-limit .value_required:n = true ,
5387   cell-space-limits .meta:n =
5388   {
5389     cell-space-top-limit = #1 ,
5390     cell-space-bottom-limit = #1 ,
5391   } ,
5392   color .tl_set:N = \l_@@_color_tl ,
5393   color .value_required:n = true ,
5394   bold .bool_set:N = \l_@@_bold_row_style_bool ,
5395   bold .default:n = true ,
5396   nb-rows .code:n =
5397   { \str_if_eq:nnTF { #1 } { * }
5398     { \int_set:Nn \l_@@_key_nb_rows_int { 500 } }
5399     { \int_set:Nn \l_@@_key_nb_rows_int { #1 } } ,
5400   nb-rows .value_required:n = true ,
5401   rowcolor .tl_set:N = \l_tmpa_tl ,
5402   rowcolor .value_required:n = true ,
5403   unknown .code:n = \@@_error:n { Unknown~key~for~RowStyle }
5404 }
```

```
5405 \NewDocumentCommand \@@_RowStyle:n { 0 { } m }
5406 {
5407   \group_begin:
5408   \tl_clear:N \l_tmpa_tl
5409   \tl_clear:N \l_@@_color_tl
5410   \int_set_eq:NN \l_@@_key_nb_rows_int \c_one_int
5411   \dim_zero:N \l_tmpa_dim
5412   \dim_zero:N \l_tmpb_dim
5413   \keys_set:nn { nicematrix / RowStyle } { #1 }
```

If the key rowcolor has been used.

```
5414   \tl_if_empty:NF \l_tmpa_tl
5415   {
```


First, the end of the current row (we remind that `\RowStyle` applies to the *end* of the current row).

```
5416 \tl_gput_right:Ne \g_@@_pre_code_before_tl
5417 {
```

The command `\@@_exp_color_arg:No` is *fully expandable*.

```
5418 \@@_exp_color_arg:No \@@_rectanglecolor \l_tmpa_tl
5419 { \int_use:N \c@iRow - \int_use:N \c@jCol }
5420 { \int_use:N \c@iRow - * }
5421 }
```

Then, the other rows (if there is several rows).

```
5422 \int_compare:nNnT \l_@@_key_nb_rows_int > \c_one_int
5423 {
5424 \tl_gput_right:Ne \g_@@_pre_code_before_tl
5425 {
5426 \@@_exp_color_arg:No \@@_rowcolor \l_tmpa_tl
5427 {
5428 \int_eval:n { \c@iRow + 1 }
5429 - \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int - 1 }
5430 }
5431 }
5432 }
5433 }
5434 \@@_put_in_row_style:n { \exp_not:n { #2 } }
```

`\l_tmpa_dim` is the value of the key `cell-space-top-limit` of `\RowStyle`.

```
5435 \dim_compare:nNnT \l_tmpa_dim > \c_zero_dim
5436 {
5437 \@@_put_in_row_style:e
5438 {
5439 \tl_gput_right:Nn \exp_not:N \g_@@_cell_after_hook_tl
5440 {
```

It's not possible to change the following code by using `\dim_set_eq:NN` (because of expansion).

```
5441 \dim_set:Nn \l_@@_cell_space_top_limit_dim
5442 { \dim_use:N \l_tmpa_dim }
5443 }
5444 }
5445 }
```

`\l_tmpb_dim` is the value of the key `cell-space-bottom-limit` of `\RowStyle`.

```
5446 \dim_compare:nNnT \l_tmpb_dim > \c_zero_dim
5447 {
5448 \@@_put_in_row_style:e
5449 {
5450 \tl_gput_right:Nn \exp_not:N \g_@@_cell_after_hook_tl
5451 {
5452 \dim_set:Nn \l_@@_cell_space_bottom_limit_dim
5453 { \dim_use:N \l_tmpb_dim }
5454 }
5455 }
5456 }
```

`\l_@@_color_tl` is the value of the key `color` of `\RowStyle`.

```
5457 \tl_if_empty:NF \l_@@_color_tl
5458 {
5459 \@@_put_in_row_style:e
5460 {
5461 \mode_leave_vertical:
5462 \@@_color:n { \l_@@_color_tl }
5463 }
5464 }
```

`\l_@@_bold_row_style_bool` is the value of the key `bold`.

```

5465   \bool_if:NT \l_@@_bold_row_style_bool
5466   {
5467     \@@_put_in_row_style:n
5468     {
5469       \exp_not:n
5470       {
5471         \if_mode_math:
5472           \c_math_toggle_token
5473           \bfseries \boldmath
5474           \c_math_toggle_token
5475         \else:
5476           \bfseries \boldmath
5477         \fi:
5478       }
5479     }
5480   }
5481   \group_end:
5482   \g_@@_row_style_tl
5483   \ignorespaces
5484 }

```

22 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).

```

5485 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e n }
5486 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e e }
5487 \cs_new_protected:Npn \@@_add_to_colors_seq:nn #1 #2
5488 {

```

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.

```

5489   \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`.

```

5490   \str_if_in:nnF { #1 } { !! }
5491   {
5492     \seq_map_indexed_inline:Nn \g_@@_colors_seq

```

```

5493         { \tl_if_eq:nnT { #1 } { ##2 } { \int_set:Nn \l_tmpa_int { ##1 } } }
5494     }
5495     \int_if_zero:nTF \l_tmpa_int

```

First, the case where the color is a *new* color (not in the sequence).

```

5496     {
5497         \seq_gput_right:Nn \g_@@_colors_seq { #1 }
5498         \tl_gset:ce { g_@@_color _ \seq_count:N \g_@@_colors_seq _ tl } { #2 }
5499     }

```

Now, the case where the color is *not* a new color (the color is in the sequence at the position `\l_tmpa_int`).

```

5500     { \tl_gput_right:ce { g_@@_color _ \int_use:N \l_tmpa_int _tl } { #2 } }
5501 }

```

The following command must be used within a `\pgfpicture`.

```

5502 \cs_new_protected:Npn \@@_clip_with_rounded_corners:
5503 {
5504     \dim_compare:nNnT \l_@@_tab_rounded_corners_dim > \c_zero_dim
5505     {

```

The TeX group is for `\pgfsetcornersarced` (whose scope is the TeX scope).

```

5506         \group_begin:
5507         \pgfsetcornersarced
5508         {
5509             \pgfpoint
5510             { \l_@@_tab_rounded_corners_dim }
5511             { \l_@@_tab_rounded_corners_dim }
5512         }

```

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.

```

5513     \bool_if:NTF \l_@@_hvlines_bool
5514     {
5515         \pgfpathrectanglecorners
5516         {
5517             \pgfpointadd
5518             { \@@_qpoint:n { row-1 } }
5519             { \pgfpoint { 0.5 \arrayrulewidth } { \c_zero_dim } }
5520         }
5521         {
5522             \pgfpointadd
5523             {
5524                 \@@_qpoint:n
5525                 { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
5526             }
5527             { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
5528         }
5529     }
5530     {
5531         \pgfpathrectanglecorners
5532         { \@@_qpoint:n { row-1 } }
5533         {
5534             \pgfpointadd
5535             {
5536                 \@@_qpoint:n
5537                 { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
5538             }
5539             { \pgfpoint \c_zero_dim \arrayrulewidth }
5540         }
5541     }

```

```

5542     \pgfusepath { clip }
5543     \group_end:

```

The TeX group was for `\pgfsetcornersarced`.

```

5544 }
5545 }

```

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`).

```

5546 \cs_new_protected:Npn \@@_actually_color:
5547 {
5548     \pgfpicture
5549     \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.

```

5550     \@@_clip_with_rounded_corners:
5551     \seq_map_indexed_inline:Nn \g_@@_colors_seq
5552     {
5553         \int_compare:nNnTF { ##1 } = \c_one_int
5554         {
5555             \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_nocolor:n
5556             \use:c { g_@@_color _ 1 _tl }
5557             \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_normal:n
5558         }
5559         {
5560             \begin { pgfscope }
5561                 \@@_color_opacity ##2
5562                 \use:c { g_@@_color _ ##1 _tl }
5563                 \tl_gclear:c { g_@@_color _ ##1 _tl }
5564                 \pgfusepath { fill }
5565                 \end { pgfscope }
5566         }
5567     }
5568     \endpgfpicture
5569 }

```

The following command will extract the potential key `opacity` in its optional argument (between square brackets) and (of course) then apply the command `\color`.

```

5570 \cs_new_protected:Npn \@@_color_opacity
5571 {
5572     \peek_meaning:NTF [
5573         { \@@_color_opacity:w }
5574         { \@@_color_opacity:w [ ] }
5575     }

```

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.

```

5576 \cs_new_protected:Npn \@@_color_opacity:w [ #1 ]
5577 {
5578     \tl_clear:N \l_tmpa_tl
5579     \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.

```

5580     \tl_if_empty:NF \l_tmpa_tl { \exp_args:No \pgfsetfillopacity \l_tmpa_tl }
5581     \tl_if_empty:NTF \l_tmpb_tl
5582     { \@declaredcolor }
5583     { \use:e { \exp_not:N \@undeclaredcolor [ \l_tmpb_tl ] } }
5584 }

```

The following set of keys is used by the command `\@@_color_opacity:wn`.

```

5585 \keys_define:nn { nicematrix / color-opacity }
5586 {
5587   opacity .tl_set:N          = \l_tmpa_tl ,
5588   opacity .value_required:n = true
5589 }

5590 \cs_new_protected:Npn \@@_cartesian_color:nn #1 #2
5591 {
5592   \cs_set_nopar:Npn \l_@@_rows_tl { #1 }
5593   \cs_set_nopar:Npn \l_@@_cols_tl { #2 }
5594   \@@_cartesian_path:
5595 }

```

Here is an example : `\@@_rowcolor {red!15} {1,3,5-7,10-}`

```

5596 \NewDocumentCommand \@@_rowcolor { 0 { } m m }
5597 {
5598   \tl_if_blank:nF { #2 }
5599   {
5600     \@@_add_to_colors_seq:en
5601     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5602     { \@@_cartesian_color:nn { #3 } { - } }
5603   }
5604 }

```

Here an example : `\@@_columncolor:nn {red!15} {1,3,5-7,10-}`

```

5605 \NewDocumentCommand \@@_columncolor { 0 { } m m }
5606 {
5607   \tl_if_blank:nF { #2 }
5608   {
5609     \@@_add_to_colors_seq:en
5610     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5611     { \@@_cartesian_color:nn { - } { #3 } }
5612   }
5613 }

```

Here is an example : `\@@_rectanglecolor{red!15}{2-3}{5-6}`

```

5614 \NewDocumentCommand \@@_rectanglecolor { 0 { } m m m }
5615 {
5616   \tl_if_blank:nF { #2 }
5617   {
5618     \@@_add_to_colors_seq:en
5619     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5620     { \@@_rectanglecolor:nnn { #3 } { #4 } { \c_zero_dim } }
5621   }
5622 }

```

The last argument is the radius of the corners of the rectangle.

```

5623 \NewDocumentCommand \@@_roundedrectanglecolor { 0 { } m m m m }
5624 {
5625   \tl_if_blank:nF { #2 }
5626   {
5627     \@@_add_to_colors_seq:en
5628     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5629     { \@@_rectanglecolor:nnn { #3 } { #4 } { #5 } }
5630   }
5631 }

```

The last argument is the radius of the corners of the rectangle.

```

5632 \cs_new_protected:Npn \@@_rectanglecolor:nnn #1 #2 #3
5633 {
5634   \@@_cut_on_hyphen:w #1 \q_stop
5635   \tl_clear_new:N \l_@@_tmpc_tl
5636   \tl_clear_new:N \l_@@_tmpd_tl
5637   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
5638   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
5639   \@@_cut_on_hyphen:w #2 \q_stop
5640   \tl_set:Ne \l_@@_rows_tl { \l_@@_tmpc_tl - \l_tmpa_tl }
5641   \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`.

```

5642   \@@_cartesian_path:n { #3 }
5643 }

```

Here is an example : `\@@_cellcolor[rgb]{0.5,0.5,0}{2-3,3-4,4-5,5-6}`

```

5644 \NewDocumentCommand \@@_cellcolor { 0 { } m m }
5645 {
5646   \clist_map_inline:nn { #3 }
5647     { \@@_rectanglecolor [ #1 ] { #2 } { ##1 } { ##1 } }
5648 }

```

```

5649 \NewDocumentCommand \@@_chessboardcolors { 0 { } m m }
5650 {
5651   \int_step_inline:nn \c@iRow
5652   {
5653     \int_step_inline:nn \c@jCol
5654     {
5655       \int_if_even:nTF { #####1 + ##1 }
5656         { \@@_cellcolor [ #1 ] { #2 } }
5657         { \@@_cellcolor [ #1 ] { #3 } }
5658       { ##1 - #####1 }
5659     }
5660   }
5661 }

```

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”.

```

5662 \NewDocumentCommand \@@_arraycolor { 0 { } m }
5663 {
5664   \@@_rectanglecolor [ #1 ] { #2 }
5665   { 1 - 1 }
5666   { \int_use:N \c@iRow - \int_use:N \c@jCol }
5667 }

5668 \keys_define:nn { nicematrix / rowcolors }
5669 {
5670   respect-blocks .bool_set:N = \l_@@_respect_blocks_bool ,
5671   respect-blocks .default:n = true ,
5672   cols .tl_set:N = \l_@@_cols_tl ,
5673   restart .bool_set:N = \l_@@_rowcolors_restart_bool ,
5674   restart .default:n = true ,
5675   unknown .code:n = \@@_error:n { Unknown~key~for~rowcolors }
5676 }

```

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*.

```
5677 \NewDocumentCommand \@@_rowlistcolors { 0 { } m m 0 { } }
5678 {
```

The group is for the options. `\l_@@_colors_seq` will be the list of colors.

```
5679 \group_begin:
5680 \seq_clear_new:N \l_@@_colors_seq
5681 \seq_set_split:Nnn \l_@@_colors_seq { , } { #3 }
5682 \tl_clear_new:N \l_@@_cols_tl
5683 \cs_set_nopar:Npn \l_@@_cols_tl { - }
5684 \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).

```
5685 \int_zero_new:N \l_@@_color_int
5686 \int_set_eq:NN \l_@@_color_int \c_one_int
5687 \bool_if:NT \l_@@_respect_blocks_bool
5688 {
```

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`).

```
5689 \seq_set_eq:NN \l_tmpb_seq \g_@@_pos_of_blocks_seq
5690 \seq_set_filter:Nnn \l_tmpa_seq \l_tmpb_seq
5691 { \@@_not_in_exterior_p:nnnn #1 }
5692 }
5693 \pgfpicture
5694 \pgf@relevantforpicturesizefalse
```

#2 is the list of intervals of rows.

```
5695 \clist_map_inline:nn { #2 }
5696 {
5697   \cs_set_nopar:Npn \l_tmpa_tl { ##1 }
5698   \tl_if_in:NnTF \l_tmpa_tl { - }
5699   { \@@_cut_on_hyphen:w ##1 \q_stop }
5700   { \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.

```
5701 \int_set:Nn \l_tmpa_int \l_tmpa_tl
5702 \int_set:Nn \l_@@_color_int
5703 { \bool_if:NNTF \l_@@_rowcolors_restart_bool 1 \l_tmpa_tl }
5704 \int_zero_new:N \l_@@_tmpc_int
5705 \int_set:Nn \l_@@_tmpc_int \l_tmpb_tl
5706 \int_do_until:nNnn \l_tmpa_int > \l_@@_tmpc_int
5707 {
```

We will compute in `\l_tmpb_int` the last row of the “block”.

```
5708 \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).

```
5709 \bool_if:NT \l_@@_respect_blocks_bool
5710 {
5711   \seq_set_filter:Nnn \l_tmpb_seq \l_tmpa_seq
5712   { \@@_intersect_our_row_p:nnnn #####1 }
5713   \seq_map_inline:Nn \l_tmpb_seq { \@@_rowcolors_i:nnnn #####1 }
```

Now, the last row of the block is computed in `\l_tmpb_int`.

```
5714 }
5715 \tl_set:No \l_@@_rows_tl
5716 { \int_use:N \l_tmpa_int - \int_use:N \l_tmpb_int }
```

`\l_@@_tmpc_tl` will be the color that we will use.

```

5717         \tl_clear_new:N \l_@@_color_tl
5718         \tl_set:Nx \l_@@_color_tl
5719         {
5720             \@@_color_index:n
5721             {
5722                 \int_mod:nn
5723                 { \l_@@_color_int - 1 }
5724                 { \seq_count:N \l_@@_colors_seq }
5725                 + 1
5726             }
5727         }
5728         \tl_if_empty:NF \l_@@_color_tl
5729         {
5730             \@@_add_to_colors_seq:ee
5731             { \tl_if_blank:nF { #1 } { [ #1 ] } { \l_@@_color_tl } }
5732             { \@@_cartesian_color:nn { \l_@@_rows_tl } { \l_@@_cols_tl } }
5733         }
5734         \int_incr:N \l_@@_color_int
5735         \int_set:Nn \l_tmpa_int { \l_tmpb_int + 1 }
5736     }
5737 }
5738 \endpgfpicture
5739 \group_end:
5740 }

```

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.

```

5741 \cs_new:Npn \@@_color_index:n #1
5742 {
5743     \str_if_eq:eeTF { \seq_item:Nn \l_@@_colors_seq { #1 } } { = }
5744     { \@@_color_index:n { #1 - 1 } }
5745     { \seq_item:Nn \l_@@_colors_seq { #1 } }
5746 }

```

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.

```

5747 \NewDocumentCommand \@@_rowcolors { 0 { } m m m }
5748 { \@@_rowlistcolors [ #1 ] { #2 } { { #3 } , { #4 } } }

```

The braces around #3 and #4 are mandatory.

```

5749 \cs_new_protected:Npn \@@_rowcolors_i:nnnnn #1 #2 #3 #4 #5
5750 {
5751     \int_compare:nNnT { #3 } > \l_tmpb_int
5752     { \int_set:Nn \l_tmpb_int { #3 } }
5753 }

5754 \prg_new_conditional:Nnn \@@_not_in_exterior:nnnnn p
5755 {
5756     \int_if_zero:nTF { #4 }
5757     \prg_return_false:
5758     {
5759         \int_compare:nNnTF { #2 } > \c@jCol
5760         \prg_return_false:
5761         \prg_return_true:
5762     }
5763 }

```


The following command return true when the block intersects the row \l_tmpa_int.

```

5764 \prg_new_conditional:Nnn \@@_intersect_our_row:nnnnn p
5765 {
5766   \int_compare:nNnTF { #1 } > \l_tmpa_int
5767     \prg_return_false:
5768     {
5769       \int_compare:nNnTF \l_tmpa_int > { #3 }
5770       \prg_return_false:
5771       \prg_return_true:
5772     }
5773 }

```

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).

```

5774 \cs_new_protected:Npn \@@_cartesian_path_normal:n #1
5775 {
5776   \dim_compare:nNnTF { #1 } = \c_zero_dim
5777   {
5778     \bool_if:NTF
5779       \l_@@_nocolor_used_bool
5780       \@@_cartesian_path_normal_ii:
5781       {
5782         \seq_if_empty:NTF \l_@@_corners_cells_seq
5783         { \@@_cartesian_path_normal_i:n { #1 } }
5784         \@@_cartesian_path_normal_ii:
5785       }
5786     }
5787   { \@@_cartesian_path_normal_i:n { #1 } }
5788 }

```

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.

```

5789 \cs_new_protected:Npn \@@_cartesian_path_normal_i:n #1
5790 {
5791   \pgfsetcornersarced { \pgfpoint { #1 } { #1 } }

```

We begin the loop over the columns.

```

5792 \clist_map_inline:Nn \l_@@_cols_tl
5793 {
5794   \cs_set_nopar:Npn \l_tmpa_tl { ##1 }
5795   \tl_if_in:NnTF \l_tmpa_tl { - }
5796     { \@@_cut_on_hyphen:w ##1 \q_stop }
5797     { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
5798   \tl_if_empty:NTF \l_tmpa_tl
5799     { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5800     {
5801       \tl_if_eq:NNT \l_tmpa_tl \c_@@_star_tl
5802       { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5803     }
5804   \tl_if_empty:NTF \l_tmpb_tl
5805     { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
5806     {
5807       \tl_if_eq:NNT \l_tmpb_tl \c_@@_star_tl
5808       { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
5809     }
5810   \int_compare:nNnT \l_tmpb_tl > \g_@@_col_total_int
5811     { \tl_set:No \l_tmpb_tl { \int_use:N \g_@@_col_total_int } }

```

`\l_@@_tmpc_tl` will contain the number of column.

```

5812 \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
5813 \@@_qpoint:n { col - \l_tmpa_tl }
5814 \int_compare:nNnTF \l_@@_first_col_int = \l_tmpa_tl
5815 { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
5816 { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
5817 \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
5818 \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }

```

We begin the loop over the rows.

```

5819 \clist_map_inline:Nn \l_@@_rows_tl
5820 {
5821   \cs_set_nopar:Npn \l_tmpa_tl { ####1 }
5822   \tl_if_in:NnTF \l_tmpa_tl { - }
5823   { \@@_cut_on_hyphen:w ####1 \q_stop }
5824   { \@@_cut_on_hyphen:w ####1 - ####1 \q_stop }
5825   \tl_if_empty:NnTF \l_tmpa_tl
5826   { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5827   {
5828     \tl_if_eq:NNT \l_tmpa_tl \c_@@_star_tl
5829     { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5830   }
5831   \tl_if_empty:NnTF \l_tmpb_tl
5832   { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }
5833   {
5834     \tl_if_eq:NNT \l_tmpb_tl \c_@@_star_tl
5835     { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }
5836   }
5837   \int_compare:nNnT \l_tmpb_tl > \g_@@_row_total_int
5838   { \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`.

```

5839 \cs_if_exist:cF
5840 { @@ _ \l_tmpa_tl _ \l_@@_tmpc_tl _ nocolor }
5841 {
5842   \@@_qpoint:n { row - \int_eval:n { \l_tmpb_tl + 1 } }
5843   \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
5844   \@@_qpoint:n { row - \l_tmpa_tl }
5845   \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
5846   \pgfpathrectanglecorners
5847   { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
5848   { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5849 }
5850 }
5851 }
5852 }

```

Now, the case where the cells will be colored cell by cell (it's mandatory for example if the key `corners` is used).

```

5853 \cs_new_protected:Npn \@@_cartesian_path_normal_ii:
5854 {
5855   \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
5856   \@@_expand_clist:NN \l_@@_rows_tl \c@iRow

```

We begin the loop over the columns.

```

5857 \clist_map_inline:Nn \l_@@_cols_tl
5858 {
5859   \@@_qpoint:n { col - ##1 }
5860   \int_compare:nNnTF \l_@@_first_col_int = { ##1 }
5861   { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
5862   { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
5863   \@@_qpoint:n { col - \int_eval:n { ##1 + 1 } }
5864   \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }

```

We begin the loop over the rows.

```

5865 \clist_map_inline:Nn \l_@@_rows_tl
5866 {
5867   \seq_if_in:NnF \l_@@_corners_cells_seq
5868   { #####1 - ##1 }
5869   {
5870     \@@_qpoint:n { row - \int_eval:n { #####1 + 1 } }
5871     \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
5872     \@@_qpoint:n { row - #####1 }
5873     \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
5874     \cs_if_exist:cF { @@ - #####1 - ##1 - nocolor }
5875     {
5876       \pgfpathrectanglecorners
5877       { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
5878       { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5879     }
5880   }
5881 }
5882 }
5883 }

```

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).

```

5884 \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

```

5885 \cs_new_protected:Npn \@@_cartesian_path_nocolor:n #1
5886 {
5887   \bool_set_true:N \l_@@_nocolor_used_bool
5888   \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
5889   \@@_expand_clist:NN \l_@@_rows_tl \c@iRow

```

We begin the loop over the columns.

```

5890 \clist_map_inline:Nn \l_@@_rows_tl
5891 {
5892   \clist_map_inline:Nn \l_@@_cols_tl
5893   { \cs_set:cpn { @@ - ##1 - #####1 - nocolor } { } }
5894 }
5895 }

```

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.

```

5896 \cs_new_protected:Npn \@@_expand_clist:NN #1 #2
5897 {
5898   \clist_set_eq:NN \l_tmpa_clist #1
5899   \clist_clear:N #1
5900   \clist_map_inline:Nn \l_tmpa_clist
5901   {
5902     \cs_set_nopar:Npn \l_tmpa_tl { ##1 }
5903     \tl_if_in:NnTF \l_tmpa_tl { - }
5904     { \@@_cut_on_hyphen:w ##1 \q_stop }
5905     { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
5906     \bool_lazy_or:nnT
5907     { \tl_if_blank_p:o \l_tmpa_tl }
5908     { \str_if_eq_p:on \l_tmpa_tl { * } }
5909     { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5910     \bool_lazy_or:nnT
5911     { \tl_if_blank_p:o \l_tmpb_tl }

```

```

5912         { \str_if_eq_p:on \l_tmpb_tl { * } }
5913         { \tl_set:No \l_tmpb_tl { \int_use:N #2 } }
5914     \int_compare:nNnT \l_tmpb_tl > #2
5915         { \tl_set:No \l_tmpb_tl { \int_use:N #2 } }
5916     \int_step_inline:nnn \l_tmpa_tl \l_tmpb_tl
5917         { \clist_put_right:Nn #1 { #####1 } }
5918 }
5919 }

```

When the user uses the key `color-inside`, the following command will be linked to `\cellcolor` in the tabular.

```

5920 \NewDocumentCommand \@@_cellcolor_tabular { 0 { } m }
5921 {
5922     \@@_test_color_inside:
5923     \tl_gput_right:Ne \g_@@_pre_code_before_tl
5924     {

```

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`).

```

5925         \@@_cellcolor [ #1 ] { \exp_not:n { #2 } }
5926         { \int_use:N \c@iRow - \int_use:N \c@jCol }
5927     }
5928     \ignorespaces
5929 }

```

When the user uses the key `color-inside`, the following command will be linked to `\rowcolor` in the tabular.

```

5930 \NewDocumentCommand \@@_rowcolor_tabular { 0 { } m }
5931 {
5932     \@@_test_color_inside:
5933     \tl_gput_right:Ne \g_@@_pre_code_before_tl
5934     {
5935         \@@_rectanglecolor [ #1 ] { \exp_not:n { #2 } }
5936         { \int_use:N \c@iRow - \int_use:N \c@jCol }
5937         { \int_use:N \c@iRow - \exp_not:n { \int_use:N \c@jCol } }
5938     }
5939     \ignorespaces
5940 }

```

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).

```

5941 \NewDocumentCommand { \@@_rowcolors_tabular } { 0 { } m m }
5942 { \@@_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.

```

5943 \NewDocumentCommand { \@@_rowlistcolors_tabular } { 0 { } m 0 { } }
5944 {
5945     \@@_test_color_inside:
5946     \peek_remove_spaces:n
5947     { \@@_rowlistcolors_tabular:nnn { #1 } { #2 } { #3 } }
5948 }

5949 \cs_new_protected:Npn \@@_rowlistcolors_tabular:nnn #1 #2 #3
5950 {

```

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`.

```

5951 \seq_gclear:N \g_tmpa_seq
5952 \seq_map_inline:Nn \g_@@_rowlistcolors_seq
5953 { \@@_rowlistcolors_tabular_i:nnnn ##1 }
5954 \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`.

```

5955 \seq_gput_right:Ne \g_@@_rowlistcolors_seq
5956 {
5957   { \int_use:N \c@iRow }
5958   { \exp_not:n { #1 } }
5959   { \exp_not:n { #2 } }
5960   { restart , cols = \int_use:N \c@jCol - , \exp_not:n { #3 } }
5961 }
5962 }

```

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`).

```

5963 \cs_new_protected:Npn \@@_rowlistcolors_tabular_i:nnnn #1 #2 #3 #4
5964 {
5965   \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).

```

5966   { \seq_gput_right:Nn \g_tmpa_seq { { #1 } { #2 } { #3 } { #4 } } }
5967   {
5968     \tl_gput_right:Ne \g_@@_pre_code_before_tl
5969     {
5970       \@@_rowlistcolors
5971       [ \exp_not:n { #2 } ]
5972       { #1 - \int_eval:n { \c@iRow - 1 } }
5973       { \exp_not:n { #3 } }
5974       [ \exp_not:n { #4 } ]
5975     }
5976   }
5977 }

```

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.

```

5978 \cs_new_protected:Npn \@@_clear_rowlistcolors_seq:
5979 {
5980   \seq_map_inline:Nn \g_@@_rowlistcolors_seq
5981   { \@@_rowlistcolors_tabular_ii:nnnn ##1 }
5982   \seq_gclear:N \g_@@_rowlistcolors_seq
5983 }

```

```

5984 \cs_new_protected:Npn \@@_rowlistcolors_tabular_ii:nnnn #1 #2 #3 #4
5985 {
5986   \tl_gput_right:Nn \g_@@_pre_code_before_tl
5987   { \@@_rowlistcolors [ #2 ] { #1 } { #3 } [ #4 ] }
5988 }

```

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.

```

5989 \NewDocumentCommand \@@_columncolor_preamble { 0 { } m }
5990 {

```

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).

```

5991   \int_compare:nNnT \c@jCol > \g_@@_col_total_int
5992   {

```

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).

```

5993     \tl_gput_left:Ne \g_@@_pre_code_before_tl
5994     {
5995       \exp_not:N \columncolor [ #1 ]
5996       { \exp_not:n { #2 } } { \int_use:N \c@jCol }
5997     }
5998   }
5999 }

```

```

6000 \hook_gput_code:nnn { begindocument } { . }
6001 {
6002   \IfPackageLoadedTF { colortbl }
6003   {
6004     \cs_set_eq:NN \@@_old_cellcolor \cellcolor
6005     \cs_set_eq:NN \@@_old_rowcolor \rowcolor
6006     \cs_new_protected:Npn \@@_revert_colortbl:
6007     {
6008       \hook_gput_code:nnn { env / tabular / begin } { nicematrix }
6009       {
6010         \cs_set_eq:NN \cellcolor \@@_old_cellcolor
6011         \cs_set_eq:NN \rowcolor \@@_old_rowcolor
6012       }
6013     }
6014   }
6015   { \cs_new_protected:Npn \@@_revert_colortbl: { } }
6016 }

```

23 The vertical and horizontal rules

OnlyMainNiceMatrix

We give to the user the possibility to define new types of columns (with `\newcolumnntype` of `array`) for special vertical rules (*e.g.* rules thicker than the standard ones) which will not extend in the potential exterior rows of the array.

We provide the command `\OnlyMainNiceMatrix` in that goal. However, that command must be no-op outside the environments of `nicematrix` (and so the user will be allowed to use the same new type of column in the environments of `nicematrix` and in the standard environments of `array`).

That's why we provide first a global definition of `\OnlyMainNiceMatrix`.

```
6017 \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`.

```
6018 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix:n #1
6019 {
6020   \int_if_zero:nTF \l_@@_first_col_int
6021     { \@@_OnlyMainNiceMatrix_i:n { #1 } }
6022     {
6023       \int_if_zero:nTF \c@jCol
6024       {
6025         \int_compare:nNnF \c@iRow = { -1 }
6026         { \int_compare:nNnF \c@iRow = { \l_@@_last_row_int - 1 } { #1 } }
6027       }
6028       { \@@_OnlyMainNiceMatrix_i:n { #1 } }
6029     }
6030 }
```

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.

```
6031 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix_i:n #1
6032 {
6033   \int_if_zero:nF \c@iRow
6034   {
6035     \int_compare:nNnF \c@iRow = \l_@@_last_row_int
6036     {
6037       \int_compare:nNnT \c@jCol > \c_zero_int
6038       { \bool_if:NF \l_@@_in_last_col_bool { #1 } }
6039     }
6040   }
6041 }
```

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.

```
6042 \keys_define:nn { nicematrix / Rules }
6043 {
6044   position .int_set:N = \l_@@_position_int ,
6045   position .value_required:n = true ,
6046   start .int_set:N = \l_@@_start_int ,
6047   end .code:n =
6048     \bool_lazy_or:nnTF
6049     { \tl_if_empty_p:n { #1 } }
6050     { \str_if_eq_p:nn { #1 } { last } }
6051     { \int_set_eq:NN \l_@@_end_int \c@jCol }
6052     { \int_set:Nn \l_@@_end_int { #1 } }
6053 }
```

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.

```
6054 \keys_define:nn { nicematrix / RulesBis }
6055 {
6056   multiplicity .int_set:N = \l_@@_multiplicity_int ,
6057   multiplicity .initial:n = 1 ,
6058   dotted .bool_set:N = \l_@@_dotted_bool ,
6059   dotted .initial:n = false ,
6060   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]`.

```
6061   color .code:n =
6062     \@@_set_CT@arc@:n { #1 }
6063     \tl_set:Nn \l_@@_rule_color_tl { #1 } ,
6064   color .value_required:n = true ,
6065   sep-color .code:n = \@@_set_CT@drsc@:n { #1 } ,
6066   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.

```
6067   tikz .code:n =
6068     \IfPackageLoadedTF { tikz }
6069     { \clist_put_right:Nn \l_@@_tikz_rule_tl { #1 } }
6070     { \@@_error:n { tikz~without~tikz } } ,
6071   tikz .value_required:n = true ,
6072   total-width .dim_set:N = \l_@@_rule_width_dim ,
6073   total-width .value_required:n = true ,
6074   width .meta:n = { total-width = #1 } ,
6075   unknown .code:n = \@@_error:n { Unknow~key~for~RulesBis }
6076 } }
```

The vertical rules

The following command will be executed in the internal `\CodeAfter`. The argument `#1` is a list of `key=value` pairs.

```
6077 \cs_new_protected:Npn \@@_vline:n #1
6078 {
```

The group is for the options.

```
6079   \group_begin:
6080   \int_set_eq:NN \l_@@_end_int \c@iRow
6081   \keys_set_known:nnN { nicematrix / Rules } { #1 } \l_@@_other_keys_tl
```

The following test is for the case where the user does not use all the columns specified in the preamble of the environment (for instance, a preamble of `|c|c|c|` but only two columns used).

```
6082   \int_compare:nNnT \l_@@_position_int < { \c@jCol + 2 }
6083     \@@_vline_i:
6084   \group_end:
6085 }

6086 \cs_new_protected:Npn \@@_vline_i:
6087 {
```

`\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`.

```
6088   \tl_set:Nn \l_tmpb_tl { \int_use:N \l_@@_position_int }
6089   \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
6090     \l_tmpa_tl
6091   {
```


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.

```

6092      \bool_gset_true:N \g_tmpa_bool
6093      \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6094        { \@@_test_vline_in_block:nnnnn ##1 }
6095      \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
6096        { \@@_test_vline_in_block:nnnnn ##1 }
6097      \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
6098        { \@@_test_vline_in_stroken_block:nnnn ##1 }
6099      \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_v:
6100      \bool_if:NTF \g_tmpa_bool
6101        {
6102          \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.

```

6103          { \int_set:Nn \l_@@_local_start_int \l_tmpa_tl }
6104        }
6105      {
6106        \int_compare:nNnT \l_@@_local_start_int > \c_zero_int
6107        {
6108          \int_set:Nn \l_@@_local_end_int { \l_tmpa_tl - 1 }
6109          \@@_vline_ii:
6110          \int_zero:N \l_@@_local_start_int
6111        }
6112      }
6113    }
6114    \int_compare:nNnT \l_@@_local_start_int > \c_zero_int
6115    {
6116      \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
6117      \@@_vline_ii:
6118    }
6119  }

```

```

6120  \cs_new_protected:Npn \@@_test_in_corner_v:
6121  {
6122    \int_compare:nNnTF \l_tmpb_tl = { \int_eval:n { \c@jCol + 1 } }
6123    {
6124      \seq_if_in:NeT
6125        \l_@@_corners_cells_seq
6126        { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
6127        { \bool_set_false:N \g_tmpa_bool }
6128    }
6129    {
6130      \seq_if_in:NeT
6131        \l_@@_corners_cells_seq
6132        { \l_tmpa_tl - \l_tmpb_tl }
6133        {
6134          \int_compare:nNnTF \l_tmpb_tl = \c_one_int
6135          { \bool_set_false:N \g_tmpa_bool }
6136          {
6137            \seq_if_in:NeT
6138              \l_@@_corners_cells_seq
6139              { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
6140              { \bool_set_false:N \g_tmpa_bool }
6141          }
6142        }
6143      }
6144    }

```

```

6145 \cs_new_protected:Npn \@@_vline_ii:
6146 {
6147   \tl_clear:N \l_@@_tikz_rule_tl
6148   \keys_set:no { nicematrix / RulesBis } \l_@@_other_keys_tl
6149   \bool_if:NTF \l_@@_dotted_bool
6150     \@@_vline_iv:
6151     {
6152       \tl_if_empty:NTF \l_@@_tikz_rule_tl
6153         \@@_vline_iii:
6154         \@@_vline_v:
6155     }
6156 }

```

First the case of a standard rule: the user has not used the key `dotted` nor the key `tikz`.

```

6157 \cs_new_protected:Npn \@@_vline_iii:
6158 {
6159   \pgfpicture
6160   \pgfrememberpicturepositiononpagetrue
6161   \pgf@relevantforpicturesizefalse
6162   \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6163   \dim_set_eq:NN \l_tmpa_dim \pgf@y
6164   \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6165   \dim_set:Nn \l_tmpb_dim
6166   {
6167     \pgf@x
6168     - 0.5 \l_@@_rule_width_dim
6169     +
6170     ( \arrayrulewidth * \l_@@_multiplicity_int
6171       + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6172   }
6173   \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6174   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
6175   \bool_lazy_all:nT
6176   {
6177     { \int_compare_p:nNn \l_@@_multiplicity_int > \c_one_int }
6178     { \cs_if_exist_p:N \CT@drsc@ }
6179     { ! \tl_if_blank_p:o \CT@drsc@ }
6180   }
6181   {
6182     \group_begin:
6183     \CT@drsc@
6184     \dim_add:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }
6185     \dim_sub:Nn \l_@@_tmpc_dim { 0.5 \arrayrulewidth }
6186     \dim_set:Nn \l_@@_tmpd_dim
6187     {
6188       \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
6189       * ( \l_@@_multiplicity_int - 1 )
6190     }
6191     \pgfpathrectanglecorners
6192     { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6193     { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
6194     \pgfusepath { fill }
6195     \group_end:
6196   }
6197   \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6198   \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
6199   \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6200   {
6201     \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
6202     \dim_sub:Nn \l_tmpb_dim \doublerulesep
6203     \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6204     \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
6205   }

```

```

6206 \CT@arc@
6207 \pgfsetlinewidth { 1.1 \arrayrulewidth }
6208 \pgfsetrectcap
6209 \pgfusepathqstroke
6210 \endpgfpicture
6211 }

```

The following code is for the case of a dotted rule (with our system of rounded dots).

```

6212 \cs_new_protected:Npn \@@_vline_iv:
6213 {
6214   \pgfpicture
6215   \pgfrememberpicturepositiononpagetrue
6216   \pgf@relevantforpicturesizefalse
6217   \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6218   \dim_set:Nn \l_@@_x_initial_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
6219   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
6220   \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6221   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
6222   \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6223   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
6224   \CT@arc@
6225   \@@_draw_line:
6226   \endpgfpicture
6227 }

```

The following code is for the case when the user uses the key `tikz`.

```

6228 \cs_new_protected:Npn \@@_vline_v:
6229 {
6230   \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).

```

6231 \CT@arc@
6232 \tl_if_empty:NF \l_@@_rule_color_tl
6233 { \tl_put_right:Ne \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6234 \pgfrememberpicturepositiononpagetrue
6235 \pgf@relevantforpicturesizefalse
6236 \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6237 \dim_set_eq:NN \l_tmpa_dim \pgf@y
6238 \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6239 \dim_set:Nn \l_tmpb_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
6240 \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6241 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
6242 \exp_args:No \tikzset \l_@@_tikz_rule_tl
6243 \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6244 ( \l_tmpb_dim , \l_tmpa_dim ) --
6245 ( \l_tmpb_dim , \l_@@_tmpc_dim ) ;
6246 \end { tikzpicture }
6247 }

```

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).

```

6248 \cs_new_protected:Npn \@@_draw_vlines:
6249 {
6250   \int_step_inline:nnn
6251   { \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool 2 1 }
6252   {
6253     \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool
6254     \c@jCol
6255     { \int_eval:n { \c@jCol + 1 } }
6256   }

```

```

6257 {
6258   \tl_if_eq:NNF \l_@@_vlines_clist \c_@@_all_tl
6259   { \clist_if_in:NnT \l_@@_vlines_clist { ##1 } }
6260   { \@@_vline:n { position = ##1 , total-width = \arrayrulewidth } }
6261 }
6262 }

```

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}`.

```

6263 \cs_new_protected:Npn \@@_hline:n #1
6264 {

```

The group is for the options.

```

6265   \group_begin:
6266   \int_zero_new:N \l_@@_end_int
6267   \int_set_eq:NN \l_@@_end_int \c@jCol
6268   \keys_set_known:nN { nicematrix / Rules } { #1 } \l_@@_other_keys_tl
6269   \@@_hline_i:
6270   \group_end:
6271 }

6272 \cs_new_protected:Npn \@@_hline_i:
6273 {
6274   \int_zero_new:N \l_@@_local_start_int
6275   \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`.

```

6276   \tl_set:N \l_tmpa_tl { \int_use:N \l_@@_position_int }
6277   \int_step_variable:nnN \l_@@_start_int \l_@@_end_int
6278   \l_tmpb_tl
6279   {

```

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.

```

6280     \bool_gset_true:N \g_tmpa_bool
6281     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6282     { \@@_test_hline_in_block:nnnnn ##1 }
6283     \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
6284     { \@@_test_hline_in_block:nnnnn ##1 }
6285     \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
6286     { \@@_test_hline_in_stroken_block:nnnn ##1 }
6287     \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_h:
6288     \bool_if:NTF \g_tmpa_bool
6289     {
6290       \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.

```

6291       { \int_set:Nn \l_@@_local_start_int \l_tmpb_tl }
6292     }
6293   {
6294     \int_compare:nNnT \l_@@_local_start_int > \c_zero_int
6295     {
6296       \int_set:Nn \l_@@_local_end_int { \l_tmpb_tl - 1 }
6297       \@@_hline_ii:
6298       \int_zero:N \l_@@_local_start_int
6299     }
6300   }
6301 }

```

```

6302     \int_compare:nNnT \l_@@_local_start_int > \c_zero_int
6303     {
6304         \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
6305         \@@_hline_ii:
6306     }
6307 }

6308 \cs_new_protected:Npn \@@_test_in_corner_h:
6309 {
6310     \int_compare:nNnTF \l_tmpa_tl = { \int_eval:n { \c@iRow + 1 } }
6311     {
6312         \seq_if_in:NeT
6313         \l_@@_corners_cells_seq
6314         { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
6315         { \bool_set_false:N \g_tmpa_bool }
6316     }
6317     {
6318         \seq_if_in:NeT
6319         \l_@@_corners_cells_seq
6320         { \l_tmpa_tl - \l_tmpb_tl }
6321         {
6322             \int_compare:nNnTF \l_tmpa_tl = \c_one_int
6323             { \bool_set_false:N \g_tmpa_bool }
6324             {
6325                 \seq_if_in:NeT
6326                 \l_@@_corners_cells_seq
6327                 { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
6328                 { \bool_set_false:N \g_tmpa_bool }
6329             }
6330         }
6331     }
6332 }

6333 \cs_new_protected:Npn \@@_hline_ii:
6334 {
6335     \tl_clear:N \l_@@_tikz_rule_tl
6336     \keys_set:no { nicematrix / RulesBis } \l_@@_other_keys_tl
6337     \bool_if:NTF \l_@@_dotted_bool
6338     \@@_hline_iv:
6339     {
6340         \tl_if_empty:NTF \l_@@_tikz_rule_tl
6341         \@@_hline_iii:
6342         \@@_hline_v:
6343     }
6344 }

```

First the case of a standard rule (without the keys dotted and tikz).

```

6345 \cs_new_protected:Npn \@@_hline_iii:
6346 {
6347     \pgfpicture
6348     \pgfrememberpicturepositiononpagetrue
6349     \pgf@relevantforpicturesizefalse
6350     \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6351     \dim_set_eq:NN \l_tmpa_dim \pgf@x
6352     \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6353     \dim_set:Nn \l_tmpb_dim
6354     {
6355         \pgf@y
6356         - 0.5 \l_@@_rule_width_dim
6357         +
6358         ( \arrayrulewidth * \l_@@_multiplicity_int

```

```

6359         + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6360     }
6361     \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6362     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
6363     \bool_lazy_all:nT
6364     {
6365         { \int_compare_p:nNn \l_@@_multiplicity_int > \c_one_int }
6366         { \cs_if_exist_p:N \CT@drsc@ }
6367         { ! \tl_if_blank_p:o \CT@drsc@ }
6368     }
6369     {
6370         \group_begin:
6371         \CT@drsc@
6372         \dim_set:Nn \l_@@_tmpd_dim
6373         {
6374             \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
6375             * ( \l_@@_multiplicity_int - 1 )
6376         }
6377         \pgfpathrectanglecorners
6378         { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6379         { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
6380         \pgfusepathqfill
6381         \group_end:
6382     }
6383     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6384     \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
6385     \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6386     {
6387         \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
6388         \dim_sub:Nn \l_tmpb_dim \doublerulesep
6389         \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6390         \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
6391     }
6392     \CT@arc@
6393     \pgfsetlinewidth { 1.1 \arrayrulewidth }
6394     \pgfsetrectcap
6395     \pgfusepathqstroke
6396     \endpgfpicture
6397 }

```

The following code is for the case of a dotted rule (with our system of rounded dots). The aim is that, by standard the dotted line fits between square brackets (`\hline` doesn't).

```

\begin{bNiceMatrix}
1 & 2 & 3 & 4 \\
\hline
1 & 2 & 3 & 4 \\
\hdottedline
1 & 2 & 3 & 4
\end{bNiceMatrix}

```

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ \hdottedline 1 & 2 & 3 & 4 \end{bmatrix}$$

But, if the user uses `margin`, the dotted line extends to have the same width as a `\hline`.

```

\begin{bNiceMatrix}[margin]
1 & 2 & 3 & 4 \\
\hline
1 & 2 & 3 & 4 \\
\hdottedline
1 & 2 & 3 & 4
\end{bNiceMatrix}

```

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ \hdottedline 1 & 2 & 3 & 4 \end{bmatrix}$$

```

6398 \cs_new_protected:Npn \@@_hline_iv:
6399 {
6400     \pgfpicture
6401     \pgfrememberpicturepositiononpagetrue

```

```

6402 \pgf@relevantforpicturesizefalse
6403 \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6404 \dim_set:Nn \l_@@_y_initial_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
6405 \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
6406 \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6407 \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
6408 \int_compare:nNnT \l_@@_local_start_int = \c_one_int
6409 {
6410   \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
6411   \bool_if:NF \g_@@_delims_bool
6412   { \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.

```

6413   \tl_if_eq:NnF \g_@@_left_delim_tl (
6414     { \dim_add:Nn \l_@@_x_initial_dim { 0.5 \l_@@_xdots_inter_dim } }
6415   )
6416 \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6417 \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
6418 \int_compare:nNnT \l_@@_local_end_int = \c_jCol
6419 {
6420   \dim_add:Nn \l_@@_x_final_dim \l_@@_right_margin_dim
6421   \bool_if:NF \g_@@_delims_bool
6422   { \dim_add:Nn \l_@@_x_final_dim \arraycolsep }
6423   \tl_if_eq:NnF \g_@@_right_delim_tl )
6424   { \dim_gsub:Nn \l_@@_x_final_dim { 0.5 \l_@@_xdots_inter_dim } }
6425 }
6426 \CT@arc@
6427 \@@_draw_line:
6428 \endpgfpicture
6429 }

```

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`).

```

6430 \cs_new_protected:Npn \@@_hline_v:
6431 {
6432   \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).

```

6433 \CT@arc@
6434 \tl_if_empty:NF \l_@@_rule_color_tl
6435 { \tl_put_right:Ne \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6436 \pgfrememberpicturepositiononpagetrue
6437 \pgf@relevantforpicturesizefalse
6438 \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6439 \dim_set_eq:NN \l_tmpa_dim \pgf@x
6440 \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6441 \dim_set:Nn \l_tmpb_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
6442 \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6443 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
6444 \exp_args:No \tikzset \l_@@_tikz_rule_tl
6445 \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6446 ( \l_tmpa_dim , \l_tmpb_dim ) --
6447 ( \l_@@_tmpc_dim , \l_tmpb_dim ) ;
6448 \end { tikzpicture }
6449 }

```

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).

```

6450 \cs_new_protected:Npn \@@_draw_hlines:
6451 {
6452   \int_step_inline:nnn
6453     { \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool 2 1 }
6454     {
6455       \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool
6456       \c@iRow
6457       { \int_eval:n { \c@iRow + 1 } }
6458     }
6459     {
6460       \tl_if_eq:NNF \l_@@_hlines_clist \c_@@_all_tl
6461       { \clist_if_in:NnT \l_@@_hlines_clist { ##1 } }
6462       { \@@_hline:n { position = ##1 , total-width = \arrayrulewidth } }
6463     }
6464   }

```

The command `\@@_Hline:` will be linked to `\Hline` in the environments of `nicematrix`.

```

6465 \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.

```

6466 \cs_set:Npn \@@_Hline_i:n #1
6467 {
6468   \peek_remove_spaces:n
6469   {
6470     \peek_meaning:NTF \Hline
6471     { \@@_Hline_ii:nn { #1 + 1 } }
6472     { \@@_Hline_iii:n { #1 } }
6473   }
6474 }
6475 \cs_set:Npn \@@_Hline_ii:nn #1 #2 { \@@_Hline_i:n { #1 } }
6476 \cs_set:Npn \@@_Hline_iii:n #1
6477 { \@@_collect_options:n { \@@_Hline_iv:nn { #1 } } }
6478 \cs_set:Npn \@@_Hline_iv:nn #1 #2
6479 {
6480   \@@_compute_rule_width:n { multiplicity = #1 , #2 }
6481   \skip_vertical:N \l_@@_rule_width_dim
6482   \tl_gput_right:Ne \g_@@_pre_code_after_tl
6483   {
6484     \@@_hline:n
6485     {
6486       multiplicity = #1 ,
6487       position = \int_eval:n { \c@iRow + 1 } ,
6488       total-width = \dim_use:N \l_@@_rule_width_dim ,
6489       #2
6490     }
6491   }
6492   \egroup
6493 }

```

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`).

```

6494 \cs_new_protected:Npn \@@_custom_line:n #1
6495 {
6496   \str_clear_new:N \l_@@_command_str
6497   \str_clear_new:N \l_@@_ccommand_str
6498   \str_clear_new:N \l_@@_letter_str
6499   \tl_clear_new:N \l_@@_other_keys_tl
6500   \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.

```

6501 \bool_lazy_all:nTF
6502 {
6503   { \str_if_empty_p:N \l_@@_letter_str }
6504   { \str_if_empty_p:N \l_@@_command_str }
6505   { \str_if_empty_p:N \l_@@_ccommand_str }
6506 }
6507 { \@@_error:n { No~letter~and~no~command } }
6508 { \@@_custom_line_i:o \l_@@_other_keys_tl }
6509 }
6510 \keys_define:nn { nicematrix / custom-line }
6511 {
6512   letter .str_set:N = \l_@@_letter_str ,
6513   letter .value_required:n = true ,
6514   command .str_set:N = \l_@@_command_str ,
6515   command .value_required:n = true ,
6516   ccommand .str_set:N = \l_@@_ccommand_str ,
6517   ccommand .value_required:n = true ,
6518 }

```

```

6519 \cs_generate_variant:Nn \@@_custom_line_i:n { o }
6520 \cs_new_protected:Npn \@@_custom_line_i:n #1
6521 {

```

The following flags will be raised when the keys `tikz`, `dotted` and `color` are used (in the `custom-line`).

```

6522 \bool_set_false:N \l_@@_tikz_rule_bool
6523 \bool_set_false:N \l_@@_dotted_rule_bool
6524 \bool_set_false:N \l_@@_color_bool
6525 \keys_set:nn { nicematrix / custom-line-bis } { #1 }
6526 \bool_if:NT \l_@@_tikz_rule_bool
6527 {
6528   \IfPackageLoadedF { tikz }
6529   { \@@_error:n { tikz-in-custom-line-without-tikz } }
6530   \bool_if:NT \l_@@_color_bool
6531   { \@@_error:n { color-in-custom-line-with-tikz } }
6532 }
6533 \bool_if:NT \l_@@_dotted_rule_bool
6534 {
6535   \int_compare:nNnT \l_@@_multiplicity_int > \c_one_int
6536   { \@@_error:n { key-multiplicity-with-dotted } }
6537 }
6538 \str_if_empty:NF \l_@@_letter_str
6539 {
6540   \int_compare:nTF { \str_count:N \l_@@_letter_str != 1 }
6541   { \@@_error:n { Several~letters } }
6542   {
6543     \tl_if_in:NoTF
6544     \c_@@_forbidden_letters_str
6545     \l_@@_letter_str
6546     { \@@_error:ne { Forbidden~letter } \l_@@_letter_str }
6547   }

```

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.

```

6548 \cs_set:cpn { @@ _ \l_@@_letter_str } ##1
6549 { \@@_v_custom_line:n { #1 } }

```

```

6550     }
6551   }
6552 }
6553 \str_if_empty:NF \l_@@_command_str { \l_@@_h_custom_line:n { #1 } }
6554 \str_if_empty:NF \l_@@_ccommand_str { \l_@@_c_custom_line:n { #1 } }
6555 }
6556 \tl_const:Nn \c_@@_forbidden_letters_tl { lcrpmbVX|()[]!@<> }
6557 \str_const:Nn \c_@@_forbidden_letters_str { lcrpmbVX|()[]!@<> }

```

The previous command `\l_@@_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.

```

6558 \keys_define:nn { nicematrix / custom-line-bis }
6559 {
6560   multiplicity .int_set:N = \l_@@_multiplicity_int ,
6561   multiplicity .initial:n = 1 ,
6562   multiplicity .value_required:n = true ,
6563   color .code:n = \bool_set_true:N \l_@@_color_bool ,
6564   color .value_required:n = true ,
6565   tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6566   tikz .value_required:n = true ,
6567   dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6568   dotted .value_forbidden:n = true ,
6569   total-width .code:n = { } ,
6570   total-width .value_required:n = true ,
6571   width .code:n = { } ,
6572   width .value_required:n = true ,
6573   sep-color .code:n = { } ,
6574   sep-color .value_required:n = true ,
6575   unknown .code:n = \@@_error:n { Unknown~key~for~custom-line }
6576 }

```

The following keys will indicate whether the keys `dotted`, `tikz` and `color` are used in the use of a `custom-line`.

```

6577 \bool_new:N \l_@@_dotted_rule_bool
6578 \bool_new:N \l_@@_tikz_rule_bool
6579 \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`.

```

6580 \keys_define:nn { nicematrix / custom-line-width }
6581 {
6582   multiplicity .int_set:N = \l_@@_multiplicity_int ,
6583   multiplicity .initial:n = 1 ,
6584   multiplicity .value_required:n = true ,
6585   tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6586   total-width .code:n = \dim_set:Nn \l_@@_rule_width_dim { #1 }
6587                       \bool_set_true:N \l_@@_total_width_bool ,
6588   total-width .value_required:n = true ,
6589   width .meta:n = { total-width = #1 } ,
6590   dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6591 }

```

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 `\l_@@_hline:n` (which is in the internal `\CodeAfter`).

```

6592 \cs_new_protected:Npn \l_@@_h_custom_line:n #1
6593 {

```

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`).

```
6594 \cs_set:cpn { nicematrix - \l_@@_command_str } { \Hline [ #1 ] }
6595 \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_command_str
6596 }
```

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`).

```
6597 \cs_new_protected:Npn \@@_c_custom_line:n #1
6598 {
```

Here, we need an expandable command since it begins with an `\noalign`.

```
6599 \exp_args:Nc \NewExpandableDocumentCommand
6600 { nicematrix - \l_@@_ccommand_str }
6601 { 0 { } m }
6602 {
6603 \noalign
6604 {
6605 \@@_compute_rule_width:n { #1 , ##1 }
6606 \skip_vertical:n { \l_@@_rule_width_dim }
6607 \clist_map_inline:nn
6608 { ##2 }
6609 { \@@_c_custom_line_i:nn { #1 , ##1 } { #####1 } }
6610 }
6611 }
6612 \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_ccommand_str
6613 }
```

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*.

```
6614 \cs_new_protected:Npn \@@_c_custom_line_i:nn #1 #2
6615 {
6616 \str_if_in:nnTF { #2 } { - }
6617 { \@@_cut_on_hyphen:w #2 \q_stop }
6618 { \@@_cut_on_hyphen:w #2 - #2 \q_stop }
6619 \tl_gput_right:Ne \g_@@_pre_code_after_tl
6620 {
6621 \@@_hline:n
6622 {
6623 #1 ,
6624 start = \l_tmpa_tl ,
6625 end = \l_tmpb_tl ,
6626 position = \int_eval:n { \c@iRow + 1 } ,
6627 total-width = \dim_use:N \l_@@_rule_width_dim
6628 }
6629 }
6630 }

6631 \cs_new_protected:Npn \@@_compute_rule_width:n #1
6632 {
6633 \bool_set_false:N \l_@@_tikz_rule_bool
6634 \bool_set_false:N \l_@@_total_width_bool
6635 \bool_set_false:N \l_@@_dotted_rule_bool
6636 \keys_set_known:nn { nicematrix / custom-line-width } { #1 }
6637 \bool_if:NF \l_@@_total_width_bool
6638 {
6639 \bool_if:NTF \l_@@_dotted_rule_bool
6640 { \dim_set:Nn \l_@@_rule_width_dim { 2 \l_@@_xdots_radius_dim } }
6641 {
6642 \bool_if:NF \l_@@_tikz_rule_bool
6643 {
```

```

6644         \dim_set:Nn \l_@@_rule_width_dim
6645         {
6646             \arrayrulewidth * \l_@@_multiplicity_int
6647             + \doublerulesep * ( \l_@@_multiplicity_int - 1 )
6648         }
6649     }
6650 }
6651 }
6652 }
6653 \cs_new_protected:Npn \@@_v_custom_line:n #1
6654 {
6655     \@@_compute_rule_width:n { #1 }

```

In the following line, the `\dim_use:N` is mandatory since we do an expansion.

```

6656     \tl_gput_right:Ne \g_@@_array_preamble_tl
6657     { \exp_not:N ! { \skip_horizontal:n { \dim_use:N \l_@@_rule_width_dim } } }
6658     \tl_gput_right:Ne \g_@@_pre_code_after_tl
6659     {
6660         \@@_vline:n
6661         {
6662             #1 ,
6663             position = \int_eval:n { \c@jCol + 1 } ,
6664             total-width = \dim_use:N \l_@@_rule_width_dim
6665         }
6666     }
6667     \@@_rec_preamble:n
6668 }
6669 \@@_custom_line:n
6670 { 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.

```

6671 \cs_new_protected:Npn \@@_test_hline_in_block:nnnnn #1 #2 #3 #4 #5
6672 {
6673     \int_compare:nNnT \l_tmpa_tl > { #1 }
6674     {
6675         \int_compare:nNnT \l_tmpa_tl < { #3 + 1 }
6676         {
6677             \int_compare:nNnT \l_tmpb_tl > { #2 - 1 }
6678             {
6679                 \int_compare:nNnT \l_tmpb_tl < { #4 + 1 }
6680                 { \bool_gset_false:N \g_tmpa_bool }
6681             }
6682         }
6683     }
6684 }

```

The same for vertical rules.

```

6685 \cs_new_protected:Npn \@@_test_vline_in_block:nnnnn #1 #2 #3 #4 #5
6686 {
6687     \int_compare:nNnT \l_tmpa_tl > { #1 - 1 }
6688     {
6689         \int_compare:nNnT \l_tmpa_tl < { #3 + 1 }
6690         {
6691             \int_compare:nNnT \l_tmpb_tl > { #2 }
6692             {
6693                 \int_compare:nNnT \l_tmpb_tl < { #4 + 1 }
6694                 { \bool_gset_false:N \g_tmpa_bool }
6695             }
6696         }
6697     }
6698 }

```

```

6696     }
6697   }
6698 }
6699 \cs_new_protected:Npn \@@_test_hline_in_stroken_block:nnnn #1 #2 #3 #4
6700 {
6701   \int_compare:nNnT \l_tmpb_tl > { #2 - 1 }
6702   {
6703     \int_compare:nNnT \l_tmpb_tl < { #4 + 1 }
6704     {
6705       \int_compare:nNnTF \l_tmpa_tl = { #1 }
6706       { \bool_gset_false:N \g_tmpa_bool }
6707       {
6708         \int_compare:nNnT \l_tmpa_tl = { #3 + 1 }
6709         { \bool_gset_false:N \g_tmpa_bool }
6710       }
6711     }
6712   }
6713 }
6714 \cs_new_protected:Npn \@@_test_vline_in_stroken_block:nnnn #1 #2 #3 #4
6715 {
6716   \int_compare:nNnT \l_tmpa_tl > { #1 - 1 }
6717   {
6718     \int_compare:nNnT \l_tmpa_tl < { #3 + 1 }
6719     {
6720       \int_compare:nNnTF \l_tmpb_tl = { #2 }
6721       { \bool_gset_false:N \g_tmpa_bool }
6722       {
6723         \int_compare:nNnT \l_tmpb_tl = { #4 + 1 }
6724         { \bool_gset_false:N \g_tmpa_bool }
6725       }
6726     }
6727   }
6728 }

```

24 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.

```

6729 \cs_new_protected:Npn \@@_compute_corners:
6730 {

```

The sequence `\l_@@_corners_cells_seq` will be the sequence of all the empty cells (and not in a block) considered in the corners of the array.

```

6731   \seq_clear_new:N \l_@@_corners_cells_seq
6732   \clist_map_inline:Nn \l_@@_corners_clist
6733   {
6734     \str_case:nnF { ##1 }
6735     {
6736       { NW }
6737       { \@@_compute_a_corner:nnnnnn 1 1 1 1 \c@iRow \c@jCol }
6738       { NE }
6739       { \@@_compute_a_corner:nnnnnn 1 \c@jCol 1 { -1 } \c@iRow 1 }
6740       { SW }
6741       { \@@_compute_a_corner:nnnnnn \c@iRow 1 { -1 } 1 1 \c@jCol }
6742       { SE }
6743       { \@@_compute_a_corner:nnnnnn \c@iRow \c@jCol { -1 } { -1 } 1 1 }
6744     }

```

```

6745     { \@@_error:nn { bad-corner } { ##1 } }
6746 }

```

Even if the user has used the key `corners` the list of cells in the corners may be empty.

```

6747 \seq_if_empty:NF \l_@@_corners_cells_seq
6748 {

```

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 color the `rows`, `columns` and `cells` must not color the cells in the corners.

```

6749     \tl_gput_right:Ne \g_@@_aux_tl
6750     {
6751         \seq_set_from_clist:Nn \exp_not:N \l_@@_corners_cells_seq
6752         { \seq_use:Nnnn \l_@@_corners_cells_seq , , , }
6753     }
6754 }
6755 }

```

“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_seq`.

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.

```

6756 \cs_new_protected:Npn \@@_compute_a_corner:nnnnnn #1 #2 #3 #4 #5 #6
6757 {

```

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.

```

6758 \bool_set_false:N \l_tmpa_bool
6759 \int_zero_new:N \l_@@_last_empty_row_int
6760 \int_set:Nn \l_@@_last_empty_row_int { #1 }
6761 \int_step_inline:nnnn { #1 } { #3 } { #5 }
6762 {
6763     \@@_test_if_cell_in_a_block:nn { ##1 } { \int_eval:n { #2 } }
6764     \bool_lazy_or:nnTF
6765     {
6766         \cs_if_exist_p:c
6767         { pgf @ sh @ ns @ \@@_env: - ##1 - \int_eval:n { #2 } }
6768     }
6769     \l_tmpb_bool
6770     { \bool_set_true:N \l_tmpa_bool }
6771     {
6772         \bool_if:NF \l_tmpa_bool
6773         { \int_set:Nn \l_@@_last_empty_row_int { ##1 } }
6774     }
6775 }

```

Now, you determine the last empty cell in the row of number 1.

```

6776 \bool_set_false:N \l_tmpa_bool
6777 \int_zero_new:N \l_@@_last_empty_column_int
6778 \int_set:Nn \l_@@_last_empty_column_int { #2 }
6779 \int_step_inline:nnnn { #2 } { #4 } { #6 }
6780 {
6781     \@@_test_if_cell_in_a_block:nn { \int_eval:n { #1 } } { ##1 }

```

```

6782 \bool_lazy_or:nnTF
6783 \l_tmpb_bool
6784 {
6785   \cs_if_exist_p:c
6786   { pgf @ sh @ ns @ \@@_env: - \int_eval:n { #1 } - ##1 }
6787 }
6788 { \bool_set_true:N \l_tmpa_bool }
6789 {
6790   \bool_if:NF \l_tmpa_bool
6791   { \int_set:Nn \l_@@_last_empty_column_int { ##1 } }
6792 }
6793 }

```

Now, we loop over the rows.

```

6794 \int_step_inline:nnnn { #1 } { #3 } \l_@@_last_empty_row_int
6795 {

```

We treat the row number ##1 with another loop.

```

6796 \bool_set_false:N \l_tmpa_bool
6797 \int_step_inline:nnnn { #2 } { #4 } \l_@@_last_empty_column_int
6798 {
6799   \@@_test_if_cell_in_a_block:nn { ##1 } { #####1 }
6800   \bool_lazy_or:nnTF
6801   \l_tmpb_bool
6802   {
6803     \cs_if_exist_p:c
6804     { pgf @ sh @ ns @ \@@_env: - ##1 - #####1 }
6805   }
6806   { \bool_set_true:N \l_tmpa_bool }
6807   {
6808     \bool_if:NF \l_tmpa_bool
6809     {
6810       \int_set:Nn \l_@@_last_empty_column_int { #####1 }
6811       \seq_put_right:Nn
6812       \l_@@_corners_cells_seq
6813       { ##1 - #####1 }
6814     }
6815   }
6816 }
6817 }
6818 }

```

The following macro tests whether a cell is in (at least) one of the blocks of the array (or in a cell with a `\diagbox`).

The flag `\l_tmpb_bool` will be raised if the cell #1-#2 is in a block (or in a cell with a `\diagbox`).

```

6819 \cs_new_protected:Npn \@@_test_if_cell_in_a_block:nn #1 #2
6820 {
6821   \int_set:Nn \l_tmpa_int { #1 }
6822   \int_set:Nn \l_tmpb_int { #2 }
6823   \bool_set_false:N \l_tmpb_bool
6824   \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6825   { \@@_test_if_cell_in_block:nnnnnn \l_tmpa_int \l_tmpb_int ##1 }
6826 }
6827 \cs_set_protected:Npn \@@_test_if_cell_in_block:nnnnnn #1 #2 #3 #4 #5 #6 #7
6828 {
6829   \int_compare:nNnF { #3 } > { #1 }
6830   {
6831     \int_compare:nNnF { #1 } > { #5 }
6832     {
6833       \int_compare:nNnF { #4 } > { #2 }
6834       {
6835         \int_compare:nNnF { #2 } > { #6 }
6836         { \bool_set_true:N \l_tmpb_bool }

```

```

6837     }
6838   }
6839 }
6840 }

```

25 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.

```

6841 \bool_new:N \l_@@_block_auto_columns_width_bool

```

Up to now, there is only one option available for the environment {NiceMatrixBlock}.

```

6842 \keys_define:nn { nicematrix / NiceMatrixBlock }
6843 {
6844   auto-columns-width .code:n =
6845   {
6846     \bool_set_true:N \l_@@_block_auto_columns_width_bool
6847     \dim_gzero_new:N \g_@@_max_cell_width_dim
6848     \bool_set_true:N \l_@@_auto_columns_width_bool
6849   }
6850 }

6851 \NewDocumentEnvironment { NiceMatrixBlock } { ! O { } }
6852 {
6853   \int_gincr:N \g_@@_NiceMatrixBlock_int
6854   \dim_zero:N \l_@@_columns_width_dim
6855   \keys_set:nn { nicematrix / NiceMatrixBlock } { #1 }
6856   \bool_if:NT \l_@@_block_auto_columns_width_bool
6857   {
6858     \cs_if_exist:cT
6859     { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
6860     {
6861       \dim_set:Nn \l_@@_columns_width_dim
6862       {
6863         \use:c
6864         { @@_max_cell_width _ \int_use:N \g_@@_NiceMatrixBlock_int }
6865       }
6866     }
6867   }
6868 }

```

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).

```

6869 {
6870   \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.

```

6871   { \int_gdecr:N \g_@@_NiceMatrixBlock_int }
6872   {
6873     \bool_if:NT \l_@@_block_auto_columns_width_bool
6874     {
6875       \iow_shipout:Nn \@mainaux \ExplSyntaxOn
6876       \iow_shipout:Ne \@mainaux
6877       {
6878         \cs_gset:cpn
6879         { @@ _ 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.

```

6880      { \dim_eval:n { \g_@@_max_cell_width_dim + \arrayrulewidth } }
6881    }
6882    \iow_shipout:Nn \@mainaux \ExplSyntaxOff
6883  }
6884 }
6885 \ignorespacesafterend
6886 }

```

26 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).

```

6887 \cs_new_protected:Npn \@@_create_extra_nodes:
6888 {
6889   \bool_if:nTF \l_@@_medium_nodes_bool
6890   {
6891     \bool_if:NTF \l_@@_large_nodes_bool
6892     \@@_create_medium_and_large_nodes:
6893     \@@_create_medium_nodes:
6894   }
6895   { \bool_if:NT \l_@@_large_nodes_bool \@@_create_large_nodes: }
6896 }

```

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`.

```

6897 \cs_new_protected:Npn \@@_computations_for_medium_nodes:
6898 {
6899   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6900   {
6901     \dim_zero_new:c { l_@@_row_\@@_i: _min_dim }
6902     \dim_set_eq:cN { l_@@_row_\@@_i: _min_dim } \c_max_dim
6903     \dim_zero_new:c { l_@@_row_\@@_i: _max_dim }
6904     \dim_set:cn { l_@@_row_\@@_i: _max_dim } { - \c_max_dim }
6905   }
6906   \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
6907   {
6908     \dim_zero_new:c { l_@@_column_\@@_j: _min_dim }
6909     \dim_set_eq:cN { l_@@_column_\@@_j: _min_dim } \c_max_dim
6910     \dim_zero_new:c { l_@@_column_\@@_j: _max_dim }
6911     \dim_set:cn { l_@@_column_\@@_j: _max_dim } { - \c_max_dim }
6912   }

```

We begin the two nested loops over the rows and the columns of the array.

```

6913   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6914   {
6915     \int_step_variable:nnNn
6916     \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.

```

6917     {
6918       \cs_if_exist:cT
6919       { 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`.

```

6920     {
6921       \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { south-west }
6922       \dim_set:cn { l_@@_row_ \@@_i: _min_dim }
6923       { \dim_min:vn { l_@@_row _ \@@_i: _min_dim } \pgf@y }
6924       \seq_if_in:NeF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
6925       {
6926         \dim_set:cn { l_@@_column _ \@@_j: _min_dim }
6927         { \dim_min:vn { l_@@_column _ \@@_j: _min_dim } \pgf@x }
6928       }

```

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`.

```

6929       \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { north-east }
6930       \dim_set:cn { l_@@_row _ \@@_i: _max_dim }
6931       { \dim_max:vn { l_@@_row _ \@@_i: _max_dim } \pgf@y }
6932       \seq_if_in:NeF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
6933       {
6934         \dim_set:cn { l_@@_column _ \@@_j: _max_dim }
6935         { \dim_max:vn { l_@@_column _ \@@_j: _max_dim } \pgf@x }
6936       }
6937     }
6938   }
6939 }

```

Now, we have to deal with empty rows or empty columns since we don't have created nodes in such rows and columns.

```

6940   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6941   {
6942     \dim_compare:nNnT
6943     { \dim_use:c { l_@@_row _ \@@_i: _min _ dim } } = \c_max_dim
6944     {
6945       \@@_qpoint:n { row - \@@_i: - base }
6946       \dim_set:cn { l_@@_row _ \@@_i: _max _ dim } \pgf@y
6947       \dim_set:cn { l_@@_row _ \@@_i: _min _ dim } \pgf@y
6948     }
6949   }
6950   \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
6951   {
6952     \dim_compare:nNnT
6953     { \dim_use:c { l_@@_column _ \@@_j: _min _ dim } } = \c_max_dim
6954     {
6955       \@@_qpoint:n { col - \@@_j: }
6956       \dim_set:cn { l_@@_column _ \@@_j: _max _ dim } \pgf@y
6957       \dim_set:cn { l_@@_column _ \@@_j: _min _ dim } \pgf@y
6958     }
6959   }
6960 }

```

Here is the command `\@@_create_medium_nodes:`. When this command is used, the “medium nodes” are created.

```

6961 \cs_new_protected:Npn \@@_create_medium_nodes:
6962 {
6963   \pgfpicture
6964     \pgfrememberpicturerepositiononpagetrue
6965     \pgf@relevantforpicturesizefalse
6966     \@@_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”.

```

6967     \cs_set_nopar:Npn \l_@@_suffix_tl { -medium }
6968     \@@_create_nodes:
6969   \endpgfpicture
6970 }

```

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:`.

```

6971 \cs_new_protected:Npn \@@_create_large_nodes:
6972 {
6973   \pgfpicture
6974     \pgfrememberpicturerepositiononpagetrue
6975     \pgf@relevantforpicturesizefalse
6976     \@@_computations_for_medium_nodes:
6977     \@@_computations_for_large_nodes:
6978     \cs_set_nopar:Npn \l_@@_suffix_tl { -large }
6979     \@@_create_nodes:
6980   \endpgfpicture
6981 }
6982 \cs_new_protected:Npn \@@_create_medium_and_large_nodes:
6983 {
6984   \pgfpicture
6985     \pgfrememberpicturerepositiononpagetrue
6986     \pgf@relevantforpicturesizefalse
6987     \@@_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”.

```

6988     \cs_set_nopar:Npn \l_@@_suffix_tl { -medium }
6989     \@@_create_nodes:
6990     \@@_computations_for_large_nodes:
6991     \cs_set_nopar:Npn \l_@@_suffix_tl { -large }
6992     \@@_create_nodes:
6993   \endpgfpicture
6994 }

```

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.

```

6995 \cs_new_protected:Npn \@@_computations_for_large_nodes:
6996 {
6997   \int_set_eq:NN \l_@@_first_row_int \c_one_int
6998   \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`.

```

6999   \int_step_variable:nNn { \c@iRow - 1 } \@@_i:
7000   {
7001     \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim }

```

¹⁴If we want to create both, we have to use `\@@_create_medium_and_large_nodes:`

```

7002     {
7003     (
7004         \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } +
7005         \dim_use:c { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
7006     )
7007     / 2
7008     }
7009     \dim_set_eq:cc { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
7010     { l_@@_row\_@@_i: _ min_dim }
7011 }
7012 \int_step_variable:nNn { \c@jCol - 1 } \@@_j:
7013 {
7014     \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim }
7015     {
7016     (
7017         \dim_use:c { l_@@_column _ \@@_j: _ max _ dim } +
7018         \dim_use:c
7019         { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
7020     )
7021     / 2
7022     }
7023     \dim_set_eq:cc { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
7024     { l_@@_column _ \@@_j: _ max _ dim }
7025 }

```

Here, we have to use `\dim_sub:cn` because of the number 1 in the name.

```

7026     \dim_sub:cn
7027     { l_@@_column _ 1 _ min _ dim }
7028     \l_@@_left_margin_dim
7029     \dim_add:cn
7030     { l_@@_column _ \int_use:N \c@jCol _ max _ dim }
7031     \l_@@_right_margin_dim
7032 }

```

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).

```

7033 \cs_new_protected:Npn \@@_create_nodes:
7034 {
7035     \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7036     {
7037         \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
7038         {

```

We draw the rectangular node for the cell (`\@@_i-\@@_j`).

```

7039         \@@_pgf_rect_node:nnnnn
7040         { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7041         { \dim_use:c { l_@@_column _ \@@_j: _ min_dim } }
7042         { \dim_use:c { l_@@_row _ \@@_i: _ min_dim } }
7043         { \dim_use:c { l_@@_column _ \@@_j: _ max_dim } }
7044         { \dim_use:c { l_@@_row _ \@@_i: _ max_dim } }
7045         \str_if_empty:NF \l_@@_name_str
7046         {
7047             \pgfnodealias
7048             { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
7049             { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7050         }
7051     }
7052 }

```

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 .

```

7053     \seq_map_pairwise_function:NNN
7054     \g_@@_multicolumn_cells_seq
7055     \g_@@_multicolumn_sizes_seq
7056     \@@_node_for_multicolumn:nn
7057 }

7058 \cs_new_protected:Npn \@@_extract_coords_values: #1 - #2 \q_stop
7059 {
7060   \cs_set_nopar:Npn \@@_i: { #1 }
7061   \cs_set_nopar:Npn \@@_j: { #2 }
7062 }

```

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”).

```

7063 \cs_new_protected:Npn \@@_node_for_multicolumn:nn #1 #2
7064 {
7065   \@@_extract_coords_values: #1 \q_stop
7066   \@@_pgf_rect_node:nnnnn
7067   { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7068   { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } }
7069   { \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } }
7070   { \dim_use:c { l_@@_column _ \int_eval:n { \@@_j: + #2 - 1 } _ max _ dim } }
7071   { \dim_use:c { l_@@_row _ \@@_i: _ max _ dim } }
7072   \str_if_empty:NF \l_@@_name_str
7073   {
7074     \pgfnodealias
7075     { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
7076     { \int_use:N \g_@@_env_int - \@@_i: - \@@_j: \l_@@_suffix_tl }
7077   }
7078 }

```

27 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).

```

7079 \keys_define:nn { nicematrix / Block / FirstPass }
7080 {
7081   j .code:n = \str_set:Nn \l_@@_hpos_block_str j
7082           \bool_set_true:N \l_@@_p_block_bool ,
7083   j .value_forbidden:n = true ,
7084   l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7085   l .value_forbidden:n = true ,
7086   r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7087   r .value_forbidden:n = true ,
7088   c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7089   c .value_forbidden:n = true ,
7090   L .code:n = \str_set:Nn \l_@@_hpos_block_str L ,
7091   L .value_forbidden:n = true ,
7092   R .code:n = \str_set:Nn \l_@@_hpos_block_str R ,

```

```

7093 R .value_forbidden:n = true ,
7094 C .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7095 C .value_forbidden:n = true ,
7096 t .code:n = \str_set:Nn \l_@@_vpos_block_str t ,
7097 t .value_forbidden:n = true ,
7098 T .code:n = \str_set:Nn \l_@@_vpos_block_str T ,
7099 T .value_forbidden:n = true ,
7100 b .code:n = \str_set:Nn \l_@@_vpos_block_str b ,
7101 b .value_forbidden:n = true ,
7102 B .code:n = \str_set:Nn \l_@@_vpos_block_str B ,
7103 B .value_forbidden:n = true ,
7104 m .code:n = \str_set:Nn \l_@@_vpos_block_str c ,
7105 m .value_forbidden:n = true ,
7106 v-center .meta:n = m ,
7107 p .code:n = \bool_set_true:N \l_@@_p_block_bool ,
7108 p .value_forbidden:n = true ,
7109 color .code:n =
7110   \@@_color:n { #1 }
7111   \tl_set_rescan:Nnn
7112     \l_@@_draw_tl
7113     { \char_set_catcode_other:N ! }
7114     { #1 } ,
7115 color .value_required:n = true ,
7116 respect-arraystretch .code:n =
7117   \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
7118 respect-arraystretch .value_forbidden:n = true ,
7119 }

```

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.

```

7120 \cs_new_protected:Npn \@@_Block: { \@@_collect_options:n { \@@_Block_i: } }

7121 \NewExpandableDocumentCommand \@@_Block_i: { m m D < > { } +m }
7122 {

```

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).

```

7123   \peek_remove_spaces:n
7124   {
7125     \tl_if_blank:nTF { #2 }
7126     { \@@_Block_ii:nnnnn \c_one_int \c_one_int }
7127     {
7128       \int_compare:nNnTF { \char_value_catcode:n { 45 } } = { 13 }
7129       \@@_Block_i_czech \@@_Block_i
7130       #2 \q_stop
7131     }
7132     { #1 } { #3 } { #4 }
7133   }
7134 }

```

With the following construction, we extract the values of i and j in the first mandatory argument of the command.

```

7135 \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`.

```

7136 {
7137   \char_set_catcode_active:N -
7138   \cs_new:Npn \@@_Block_i_czech #1-#2 \q_stop { \@@_Block_ii:nnnnn { #1 } { #2 } }
7139 }

```

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.

```
7140 \cs_new_protected:Npn \@@_Block_ii:nnnnn #1 #2 #3 #4 #5
7141 {
```

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).

```
7142   \bool_lazy_or:nnTF
7143   { \tl_if_blank_p:n { #1 } }
7144   { \str_if_eq_p:Vn \c_@@_star_str { #1 } }
7145   { \int_set:Nn \l_tmpa_int { 100 } }
7146   { \int_set:Nn \l_tmpa_int { #1 } }
7147   \bool_lazy_or:nnTF
7148   { \tl_if_blank_p:n { #2 } }
7149   { \str_if_eq_p:Vn \c_@@_star_str { #2 } }
7150   { \int_set:Nn \l_tmpb_int { 100 } }
7151   { \int_set:Nn \l_tmpb_int { #2 } }
```

If the block is mono-column.

```
7152   \int_compare:nNnTF \l_tmpb_int = \c_one_int
7153   {
7154     \tl_if_empty:NTF \l_@@_hpos_cell_tl
7155     { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_c_str }
7156     { \str_set:NV \l_@@_hpos_block_str \l_@@_hpos_cell_tl }
7157   }
7158   { \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.

```
7159   \keys_set_known:nn { nicematrix / Block / FirstPass } { #3 }
7160   \tl_set:Ne \l_tmpa_tl
7161   {
7162     { \int_use:N \c@iRow }
7163     { \int_use:N \c@jCol }
7164     { \int_eval:n { \c@iRow + \l_tmpa_int - 1 } }
7165     { \int_eval:n { \c@jCol + \l_tmpb_int - 1 } }
7166   }
```

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).

```
7167   \bool_set_false:N \l_tmpa_bool
7168   \bool_if:NT \l_@@_amp_in_blocks_bool
7169   { \tl_if_in:nnT { #5 } { & } { \bool_set_true:N \l_tmpa_bool } }
7170   \bool_case:nF
7171   {
7172     \l_tmpa_bool { \@@_Block_vii:eennn }
7173     \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.

```

7174         \l_@@_X_bool                               { \@@_Block_v:eennn }
7175         { \tl_if_empty_p:n { #5 } }                  { \@@_Block_v:eennn }
7176         { \int_compare_p:nNn \l_tmpa_int = \c_one_int } { \@@_Block_iv:eennn }
7177         { \int_compare_p:nNn \l_tmpb_int = \c_one_int } { \@@_Block_iv:eennn }
7178     }
7179     { \@@_Block_v:eennn }
7180     { \l_tmpa_int } { \l_tmpb_int } { #3 } { #4 } { #5 }
7181 }

```

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.

```

7182 \cs_generate_variant:Nn \@@_Block_iv:nnnnn { e e n n n }
7183 \cs_new_protected:Npn \@@_Block_iv:nnnnn #1 #2 #3 #4 #5
7184 {
7185     \int_gincr:N \g_@@_block_box_int
7186     \cs_set_protected_nopar:Npn \diagbox ##1 ##2
7187     {
7188         \tl_gput_right:Ne \g_@@_pre_code_after_tl
7189         {
7190             \@@_actually_diagbox:nnnnnn
7191             { \int_use:N \c@iRow }
7192             { \int_use:N \c@jCol }
7193             { \int_eval:n { \c@iRow + #1 - 1 } }
7194             { \int_eval:n { \c@jCol + #2 - 1 } }
7195             { \g_@@_row_style_tl \exp_not:n { ##1 } }
7196             { \g_@@_row_style_tl \exp_not:n { ##2 } }
7197         }
7198     }
7199     \box_gclear_new:c
7200     { 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!

```

7201 \hbox_gset:cn
7202 { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7203 {

```

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}`).

```

7204 \tl_if_empty:NTF \l_@@_color_tl
7205 { \int_compare:nNnT { #2 } = \c_one_int \set@color }
7206 { \@@_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`.

```

7207 \int_compare:nNnT { #1 } = \c_one_int
7208 {
7209     \int_if_zero:nTF \c@iRow
7210     {

```


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`.

```

$\begin{bNiceMatrix}%
[
  r,
  first-row,
  last-col,
  code-for-first-row = \Block{}{\scriptstyle\color{blue} \arabic{jCol}},
  code-for-last-col = \scriptstyle \color{blue} \arabic{iRow}
]
& & & & \\
-2 & 3 & -4 & 5 & \\
3 & -4 & 5 & -6 & \\
-4 & 5 & -6 & 7 & \\
5 & -6 & 7 & -8 & \\
\end{bNiceMatrix}$

```

```

7211         \cs_set_eq:NN \Block \@@_NullBlock:
7212         \l_@@_code_for_first_row_tl
7213       }
7214       {
7215         \int_compare:nNnT \c@iRow = \l_@@_last_row_int
7216         {
7217           \cs_set_eq:NN \Block \@@_NullBlock:
7218           \l_@@_code_for_last_row_tl
7219         }
7220       }
7221       \g_@@_row_style_tl
7222     }

```

The following command will be no-op when `respect-arraystretch` is in force.

```

7223     \@@_reset_arraystretch:
7224     \dim_zero:N \extrarowheight

```

#4 is the optional argument of the command `\Block`, provided with the syntax `<...>`.

```

7225     #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.).

```

7226     \@@_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}`.

```

7227     \bool_if:NTF \l_@@_tabular_bool
7228     {
7229       \bool_lazy_all:nTF
7230       {
7231         { \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.

```

7232         { ! \dim_compare_p:nNn \l_@@_col_width_dim < \c_zero_dim }
7233         { ! \g_@@_rotate_bool }
7234       }

```

When the block is mono-column in a column with a fixed width (e.g. `p{3cm}`), we use a `{minipage}`.

```

7235         {
7236         \use:e
7237         {
The \exp_not:N is mandatory before \begin.
7238         \exp_not:N \begin { minipage }%
7239         [ \str_lowercase:o \l_@@_vpos_block_str ]
7240         { \l_@@_col_width_dim }
7241         \str_case:on \l_@@_hpos_block_str
7242         { c \centering r \raggedleft l \raggedright }
7243         }
7244         #5
7245         \end { minipage }
7246     }

```

In the other cases, we use a `{tabular}`.

```

7247     {
7248     \use:e
7249     {
7250     \exp_not:N \begin { tabular }%
7251     [ \str_lowercase:o \l_@@_vpos_block_str ]
7252     { @ { } \l_@@_hpos_block_str @ { } }
7253     }
7254     #5
7255     \end { tabular }
7256     }
7257 }

```

If we are in a mathematical array (`\l_@@_tabular_bool` is false). The composition is always done with an `{array}` (never with a `{minipage}`).

```

7258     {
7259     \c_math_toggle_token
7260     \use:e
7261     {
7262     \exp_not:N \begin { array }%
7263     [ \str_lowercase:o \l_@@_vpos_block_str ]
7264     { @ { } \l_@@_hpos_block_str @ { } }
7265     }
7266     #5
7267     \end { array }
7268     \c_math_toggle_token
7269     }
7270 }

```

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).

```

7271 \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.

```

7272 \int_compare:nNtT { #2 } = \c_one_int
7273 {
7274     \dim_gset:Nn \g_@@_blocks_wd_dim
7275     {
7276     \dim_max:nn
7277     \g_@@_blocks_wd_dim
7278     {
7279     \box_wd:c
7280     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7281     }
7282     }
7283 }

```

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.

```

7284     \bool_lazy_and:nnT
7285     { \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.

```

7286     { \str_if_empty_p:N \l_@@_vpos_block_str }
7287     {
7288         \dim_gset:Nn \g_@@_blocks_ht_dim
7289         {
7290             \dim_max:nn
7291             \g_@@_blocks_ht_dim
7292             {
7293                 \box_ht:c
7294                 { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7295             }
7296         }
7297         \dim_gset:Nn \g_@@_blocks_dp_dim
7298         {
7299             \dim_max:nn
7300             \g_@@_blocks_dp_dim
7301             {
7302                 \box_dp:c
7303                 { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7304             }
7305         }
7306     }
7307     \seq_gput_right:Ne \g_@@_blocks_seq
7308     {
7309         \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.

```

7310     {
7311         \exp_not:n { #3 } ,
7312         \l_@@_hpos_block_str ,

```

Now, we put a key for the vertical alignment.

```

7313         \bool_if:NT \g_@@_rotate_bool
7314         {
7315             \bool_if:NTF \g_@@_rotate_c_bool
7316             { m }
7317             { \int_compare:nNnT \c@iRow = \l_@@_last_row_int T }
7318         }
7319     }
7320     {
7321         \box_use_drop:c
7322         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7323     }
7324 }
7325 \bool_set_false:N \g_@@_rotate_c_bool
7326 }

```

```

7327 \cs_new:Npn \@@_adjust_hpos_rotate:
7328 {
7329     \bool_if:NT \g_@@_rotate_bool
7330     {
7331         \str_set:Ne \l_@@_hpos_block_str
7332         {

```

```

7333         \bool_if:NTF \g_@@_rotate_c_bool
7334         { c }
7335         {
7336             \str_case:onF \l_@@_vpos_block_str
7337             { b l B l t r T r }
7338             { \int_compare:nNnTF \c@iRow = \l_@@_last_row_int r l }
7339         }
7340     }
7341 }
7342 }

```

Despite its name the following command rotates the box of the block *but also does vertical adjustment of the baseline of the block*.

```

7343 \cs_new_protected:Npn \@@_rotate_box_of_block:
7344 {
7345     \box_grotate:cn
7346     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7347     { 90 }
7348     \int_compare:nNnT \c@iRow = \l_@@_last_row_int
7349     {
7350         \vbox_gset_top:cn
7351         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7352         {
7353             \skip_vertical:n { 0.8 ex }
7354             \box_use:c
7355             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7356         }
7357     }
7358     \bool_if:NT \g_@@_rotate_c_bool
7359     {
7360         \hbox_gset:cn
7361         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7362         {
7363             \c_math_toggle_token
7364             \vcenter
7365             {
7366                 \box_use:c
7367                 { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7368             }
7369             \c_math_toggle_token
7370         }
7371     }
7372 }

```

The following macro is for the standard case, where the block is not mono-row and not mono-column and does not use the key p). In that case, the content of the block is *not* composed right now in a box. The composition in a box will be done further, just after the construction of the array (cf. `\@@_draw_blocks:` and above all `\@@_Block_v:nnnnnn`).

#1 is i (the number of rows of the block), #2 is j (the number of columns of the block), #3 is the list of *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.

```

7373 \cs_generate_variant:Nn \@@_Block_v:nnnnn { e e n n n }
7374 \cs_new_protected:Npn \@@_Block_v:nnnnn #1 #2 #3 #4 #5
7375 {
7376     \seq_gput_right:Ne \g_@@_blocks_seq
7377     {
7378         \l_tmpa_tl
7379         { \exp_not:n { #3 } }
7380         {
7381             \bool_if:NTF \l_@@_tabular_bool
7382             {
7383                 \group_begin:

```

The following command will be no-op when `respect-arraystretch` is in force.

```

7384         \@@_reset_arraystretch:
7385         \exp_not:n
7386         {
7387             \dim_zero:N \extrarowheight
7388             #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).

```

7389         \bool_if:NT \c_@@_testphase_table_bool
7390         { \tag_stop:n { table } }
7391         \use:e
7392         {
7393             \exp_not:N \begin { tabular } [ \l_@@_vpos_block_str ]
7394             { @ { } \l_@@_hpos_block_str @ { } }
7395         }
7396         #5
7397         \end { tabular }
7398     }
7399     \group_end:
7400 }

```

When we are *not* in an environment `{NiceTabular}` (or similar).

```

7401     {
7402         \group_begin:

```

The following will be no-op when `respect-arraystretch` is in force.

```

7403         \@@_reset_arraystretch:
7404         \exp_not:n
7405         {
7406             \dim_zero:N \extrarowheight
7407             #4
7408             \c_math_toggle_token
7409             \use:e
7410             {
7411                 \exp_not:N \begin { array } [ \l_@@_vpos_block_str ]
7412                 { @ { } \l_@@_hpos_block_str @ { } }
7413             }
7414             #5
7415             \end { array }
7416             \c_math_toggle_token
7417         }
7418         \group_end:
7419     }
7420 }
7421 }
7422 }

```

The following macro is for the case of a `\Block` which uses the key `p`.

```

7423 \cs_generate_variant:Nn \@@_Block_vi:nnnnn { e e n n n }
7424 \cs_new_protected:Npn \@@_Block_vi:nnnnn #1 #2 #3 #4 #5
7425 {
7426     \seq_gput_right:Ne \g_@@_blocks_seq
7427     {
7428         \l_tmpa_tl
7429         { \exp_not:n { #3 } }
7430         {
7431             \group_begin:
7432             \exp_not:n { #4 #5 }
7433             \group_end:
7434         }

```

```

7435     }
7436 }

```

The following macro is for the case of a `\Block` which uses the key `p`.

```

7437 \cs_generate_variant:Nn \@@_Block_vii:nnnnn { e e n n n }
7438 \cs_new_protected:Npn \@@_Block_vii:nnnnn #1 #2 #3 #4 #5
7439 {
7440   \seq_gput_right:Ne \g_@@_blocks_seq
7441   {
7442     \l_tmpa_tl
7443     { \exp_not:n { #3 } }
7444     { \exp_not:n { #4 #5 } }
7445   }
7446 }

```

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).

```

7447 \keys_define:nn { nicematrix / Block / SecondPass }
7448 {
7449   ampersand-in-blocks .bool_set:N = \l_@@_amp_in_blocks_bool ,
7450   ampersand-in-blocks .default:n = true ,
7451   &-in-blocks .meta:n = ampersand-in-blocks ,
7452   tikz .code:n =
7453     \IfPackageLoadedTF { tikz }
7454     { \seq_put_right:Nn \l_@@_tikz_seq { { #1 } } }
7455     { \@@_error:n { tikz~key~without~tikz } } ,
7456   tikz .value_required:n = true ,
7457   fill .code:n =
7458     \tl_set_rescan:Nnn
7459     \l_@@_fill_tl
7460     { \char_set_catcode_other:N ! }
7461     { #1 } ,
7462   fill .value_required:n = true ,
7463   opacity .tl_set:N = \l_@@_opacity_tl ,
7464   opacity .value_required:n = true ,
7465   draw .code:n =
7466     \tl_set_rescan:Nnn
7467     \l_@@_draw_tl
7468     { \char_set_catcode_other:N ! }
7469     { #1 } ,
7470   draw .default:n = default ,
7471   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
7472   rounded-corners .default:n = 4 pt ,
7473   color .code:n =
7474     \@@_color:n { #1 }
7475     \tl_set_rescan:Nnn
7476     \l_@@_draw_tl
7477     { \char_set_catcode_other:N ! }
7478     { #1 } ,
7479   borders .clist_set:N = \l_@@_borders_clist ,
7480   borders .value_required:n = true ,
7481   hvlines .meta:n = { vlines , hlines } ,
7482   vlines .bool_set:N = \l_@@_vlines_block_bool ,
7483   vlines .default:n = true ,
7484   hlines .bool_set:N = \l_@@_hlines_block_bool ,
7485   hlines .default:n = true ,
7486   line-width .dim_set:N = \l_@@_line_width_dim ,
7487   line-width .value_required:n = true ,

```

Some keys have not a property `.value_required:n` (or similar) because they are in `FirstPass`.

```

7488   j .code:n = \str_set:Nn \l_@@_hpos_block_str j

```

```

7489         \bool_set_true:N \l_@@_p_block_bool ,
7490     l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7491     r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7492     c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7493     L .code:n = \str_set:Nn \l_@@_hpos_block_str l
7494         \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7495     R .code:n = \str_set:Nn \l_@@_hpos_block_str r
7496         \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7497     C .code:n = \str_set:Nn \l_@@_hpos_block_str c
7498         \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7499     t .code:n = \str_set:Nn \l_@@_vpos_block_str t ,
7500     T .code:n = \str_set:Nn \l_@@_vpos_block_str T ,
7501     b .code:n = \str_set:Nn \l_@@_vpos_block_str b ,
7502     B .code:n = \str_set:Nn \l_@@_vpos_block_str B ,
7503     m .code:n = \str_set:Nn \l_@@_vpos_block_str c ,
7504     m .value_forbidden:n = true ,
7505     v-center .meta:n = m ,
7506     p .code:n = \bool_set_true:N \l_@@_p_block_bool ,
7507     p .value_forbidden:n = true ,
7508     name .tl_set:N = \l_@@_block_name_str ,
7509     name .value_required:n = true ,
7510     name .initial:n = ,
7511     respect-arraystretch .code:n =
7512         \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
7513     respect-arraystretch .value_forbidden:n = true ,
7514     transparent .bool_set:N = \l_@@_transparent_bool ,
7515     transparent .default:n = true ,
7516     transparent .initial:n = false ,
7517     unknown .code:n = \@@_error:n { Unknown-key-for-Block }
7518 }

```

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.

```

7519 \cs_new_protected:Npn \@@_draw_blocks:
7520 {
7521     \bool_if:NTF \c_@@_tagging_array_bool
7522     { \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign: }
7523     { \cs_set_eq:NN \ialign \@@_old_ialign: }
7524     \seq_map_inline:Nn \g_@@_blocks_seq { \@@_Block_iv:nnnnnn ##1 }
7525 }
7526 \cs_generate_variant:Nn \@@_Block_v:nnnnnn { n n V n n }
7527 \cs_new_protected:Npn \@@_Block_iv:nnnnnn #1 #2 #3 #4 #5 #6
7528 {

```

The integer `\l_@@_last_row_int` will be the last row of the block and `\l_@@_last_col_int` its last column.

```

7529     \int_zero_new:N \l_@@_last_row_int
7530     \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.

```

7531     \int_compare:nNnTF { #3 } > { 99 }
7532     { \int_set_eq:NN \l_@@_last_row_int \c@iRow }
7533     { \int_set:Nn \l_@@_last_row_int { #3 } }
7534     \int_compare:nNnTF { #4 } > { 99 }
7535     { \int_set_eq:NN \l_@@_last_col_int \c@jCol }
7536     { \int_set:Nn \l_@@_last_col_int { #4 } }

```

```

7537 \int_compare:nNnTF \l_@@_last_col_int > \g_@@_col_total_int
7538 {
7539   \bool_lazy_and:nnTF
7540     \l_@@_preamble_bool
7541     {
7542       \int_compare_p:n
7543         { \l_@@_last_col_int <= \g_@@_static_num_of_col_int }
7544     }
7545     {
7546       \msg_error:nnnn { nicematrix } { Block-too-large-2 } { #1 } { #2 }
7547       \@@_msg_redirect_name:nn { Block-too-large-2 } { none }
7548       \@@_msg_redirect_name:nn { columns-not-used } { none }
7549     }
7550     { \msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 } }
7551   }
7552   {
7553     \int_compare:nNnTF \l_@@_last_row_int > \g_@@_row_total_int
7554     { \msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 } }
7555     {
7556       \@@_Block_v:nnVVnn
7557         { #1 }
7558         { #2 }
7559         \l_@@_last_row_int
7560         \l_@@_last_col_int
7561         { #5 }
7562         { #6 }
7563     }
7564   }
7565 }

```

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

```

7566 \cs_new_protected:Npn \@@_Block_v:nnnnnn #1 #2 #3 #4 #5 #6
7567 {

```

The group is for the keys.

```

7568   \group_begin:
7569   \int_compare:nNnT { #1 } = { #3 }
7570   { \str_set:Nn \l_@@_vpos_block_str { t } }
7571   \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).

```

7572   \tl_if_in:nnT { #6 } { & } { \bool_set_true:N \l_@@_ampersand_bool }
7573   \bool_lazy_and:nnT
7574     \l_@@_vlines_block_bool
7575     { ! \l_@@_ampersand_bool }
7576   {
7577     \tl_gput_right:Ne \g_nicematrix_code_after_tl
7578     {
7579       \@@_vlines_block:nnn
7580       { \exp_not:n { #5 } }
7581       { #1 - #2 }
7582       { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7583     }
7584   }
7585   \bool_if:NT \l_@@_hlines_block_bool
7586   {
7587     \tl_gput_right:Ne \g_nicematrix_code_after_tl
7588     {
7589       \@@_hlines_block:nnn
7590       { \exp_not:n { #5 } }
7591       { #1 - #2 }

```



```

7592         { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7593     }
7594 }
7595 \bool_if:NF \l_@@_transparent_bool
7596 {
7597     \bool_lazy_and:nnF \l_@@_vlines_block_bool \l_@@_hlines_block_bool
7598     {

```

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).

```

7599         \seq_gput_left:Ne \g_@@_pos_of_blocks_seq
7600         { { #1 } { #2 } { #3 } { #4 } { \l_@@_block_name_str } }
7601     }
7602 }

7603 \tl_if_empty:NF \l_@@_draw_tl
7604 {
7605     \bool_lazy_or:nnT \l_@@_hlines_block_bool \l_@@_vlines_block_bool
7606     { \@@_error:n { hlines~with~color } }
7607     \tl_gput_right:Ne \g_nicematrix_code_after_tl
7608     {
7609         \@@_stroke_block:nnn

```

#5 are the options

```

7610         { \exp_not:n { #5 } }
7611         { #1 - #2 }
7612         { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7613     }
7614     \seq_gput_right:Nn \g_@@_pos_of_stroken_blocks_seq
7615     { { #1 } { #2 } { #3 } { #4 } }
7616 }

7617 \clist_if_empty:NF \l_@@_borders_clist
7618 {
7619     \tl_gput_right:Ne \g_nicematrix_code_after_tl
7620     {
7621         \@@_stroke_borders_block:nnn
7622         { \exp_not:n { #5 } }
7623         { #1 - #2 }
7624         { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7625     }
7626 }

7627 \tl_if_empty:NF \l_@@_fill_tl
7628 {
7629     \tl_if_empty:NF \l_@@_opacity_tl
7630     {
7631         \tl_if_head_eq_meaning:nNTF \l_@@_fill_tl [
7632             {
7633                 \tl_set:Ne \l_@@_fill_tl
7634                 {
7635                     [ opacity = \l_@@_opacity_tl ,
7636                     \tl_tail:o \l_@@_fill_tl
7637                 }
7638             }
7639             {
7640                 \tl_set:Ne \l_@@_fill_tl
7641                 { [ opacity = \l_@@_opacity_tl ] { \l_@@_fill_tl } }
7642             }
7643         ]
7644     }
7645     \tl_gput_right:Ne \g_@@_pre_code_before_tl
7646     {
7647         \exp_not:N \roundedrectanglecolor
7648         \tl_if_head_eq_meaning:oNTF \l_@@_fill_tl [

```

```

7648         { \l_@@_fill_tl }
7649         { { \l_@@_fill_tl } }
7650     { #1 - #2 }
7651     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7652     { \dim_use:N \l_@@_rounded_corners_dim }
7653 }
7654 }
7655 \seq_if_empty:NF \l_@@_tikz_seq
7656 {
7657     \tl_gput_right:Ne \g_nicematrix_code_before_tl
7658     {
7659         \@@_block_tikz:nnnnn
7660         { #1 }
7661         { #2 }
7662         { \int_use:N \l_@@_last_row_int }
7663         { \int_use:N \l_@@_last_col_int }
7664         { \seq_use:Nn \l_@@_tikz_seq { , } }
7665     }
7666 }

7667 \cs_set_protected_nopar:Npn \diagbox ##1 ##2
7668 {
7669     \tl_gput_right:Ne \g_@@_pre_code_after_tl
7670     {
7671         \@@_actually_diagbox:nnnnnn
7672         { #1 }
7673         { #2 }
7674         { \int_use:N \l_@@_last_row_int }
7675         { \int_use:N \l_@@_last_col_int }
7676         { \exp_not:n { ##1 } } { \exp_not:n { ##2 } }
7677     }
7678 }

```

Let's consider the following `{NiceTabular}`. Because of the instruction `!\hspace{1cm}` in the preamble which increases the space between the columns (by adding, in fact, that space to the previous column, that is to say the second column of the tabular), we will create *two* nodes relative to the block: the node `1-1-block` and the node `1-1-block-short`.

```

\begin{NiceTabular}{cc!\hspace{1cm}}c}
\Block{2-2}{our block} &      & one      & \\
                        &      & two      & \\
three                  & four & five     & \\
six                    & seven & eight    & \\
\end{NiceTabular}

```

We highlight the node `1-1-block`

our block		one
three	four	two
six	seven	five
		eight

We highlight the node `1-1-block-short`

our block		one
three	four	two
six	seven	five
		eight

The construction of the node corresponding to the merged cells.

```

7679 \pgfpicture
7680 \pgfrememberpicturepositiononpagetrue
7681 \pgf@relevantforpicturesizefalse
7682 \@@_qpoint:n { row - #1 }
7683 \dim_set_eq:NN \l_tmpa_dim \pgf@y
7684 \@@_qpoint:n { col - #2 }
7685 \dim_set_eq:NN \l_tmpb_dim \pgf@x
7686 \@@_qpoint:n { row - \int_eval:n { \l_@@_last_row_int + 1 } }
7687 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y

```

```

7688 \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7689 \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.

```

7690 \@@_pgf_rect_node:nnnnn
7691 { \@@_env: - #1 - #2 - block }
7692 \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
7693 \str_if_empty:NF \l_@@_block_name_str
7694 {
7695   \pgfnodealias
7696   { \@@_env: - \l_@@_block_name_str }
7697   { \@@_env: - #1 - #2 - block }
7698   \str_if_empty:NF \l_@@_name_str
7699   {
7700     \pgfnodealias
7701     { \l_@@_name_str - \l_@@_block_name_str }
7702     { \@@_env: - #1 - #2 - block }
7703   }
7704 }

```

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.

```

7705 \bool_if:NF \l_@@_hpos_of_block_cap_bool
7706 {
7707   \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.

```

7708 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
7709 {

```

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.

```

7710 \cs_if_exist:cT
7711 { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
7712 {
7713   \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
7714   {
7715     \pgfpointanchor { \@@_env: - ##1 - #2 } { west }
7716     \dim_set:Nn \l_tmpb_dim { \dim_min:nn \l_tmpb_dim \pgf@x }
7717   }
7718 }
7719 }

```

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.

```

7720 \dim_compare:nNnT \l_tmpb_dim = \c_max_dim
7721 {
7722   \@@_qpoint:n { col - #2 }
7723   \dim_set_eq:NN \l_tmpb_dim \pgf@x
7724 }
7725 \dim_set:Nn \l_@@_tmpd_dim { - \c_max_dim }
7726 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
7727 {
7728   \cs_if_exist:cT
7729   { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
7730   {
7731     \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
7732     {
7733       \pgfpointanchor

```

```

7734         { \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
7735         { east }
7736         \dim_set:Nn \l_@@_tmpd_dim { \dim_max:nn \l_@@_tmpd_dim \pgf@x }
7737     }
7738 }
7739 }
7740 \dim_compare:nNnT \l_@@_tmpd_dim = { - \c_max_dim }
7741 {
7742     \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7743     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
7744 }
7745 \@@_pgf_rect_node:nnnn
7746 { \@@_env: - #1 - #2 - block - short }
7747 \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
7748 }

```

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.

```

7749 \bool_if:NT \l_@@_medium_nodes_bool
7750 {
7751     \@@_pgf_rect_node:nnn
7752     { \@@_env: - #1 - #2 - block - medium }
7753     { \pgfpointanchor { \@@_env: - #1 - #2 - medium } { north-west } }
7754     {
7755         \pgfpointanchor
7756         { \@@_env:
7757             - \int_use:N \l_@@_last_row_int
7758             - \int_use:N \l_@@_last_col_int - medium
7759         }
7760         { south-east }
7761     }
7762 }
7763 \endpgfpicture

```

```

7764 \bool_if:NTF \l_@@_ampersand_bool
7765 {
7766     \seq_set_split:Nnn \l_tmpa_seq { & } { #6 }
7767     \int_zero_new:N \l_@@_split_int
7768     \int_set:Nn \l_@@_split_int { \seq_count:N \l_tmpa_seq }
7769     \pgfpicture
7770     \pgfrememberpicturepositiononpagetrue
7771     \pgf@relevantforpicturesizefalse
7772     \@@_qpoint:n { row - #1 }
7773     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7774     \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
7775     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
7776     \@@_qpoint:n { col - #2 }
7777     \dim_set_eq:NN \l_tmpa_dim \pgf@x
7778     \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
7779     \dim_set:Nn \l_tmpb_dim
7780     { ( \pgf@x - \l_tmpa_dim ) / \int_use:N \l_@@_split_int }
7781     \bool_lazy_or:nnT
7782     \l_@@_vlines_block_bool
7783     { \tl_if_eq_p:NN \l_@@_vlines_clist \c_@@_all_tl }
7784     {
7785         \int_step_inline:nn { \l_@@_split_int - 1 }
7786         {
7787             \pgfpathmoveto
7788             {
7789                 \pgfpoint
7790                 { \l_tmpa_dim + ##1 \l_tmpb_dim }
7791                 \l_@@_tmpc_dim
7792             }

```

```

7793         \pgfpathlineto
7794         {
7795             \pgfpoint
7796             { \l_tmpa_dim + ##1 \l_tmpb_dim }
7797             \l_@@_tmpd_dim
7798         }
7799         \CT@arc@
7800         \pgfsetlinewidth { 1.1 \arrayrulewidth }
7801         \pgfsetrectcap
7802         \pgfusepathqstroke
7803     }
7804 }
7805 \@@_qpoint:n { row - #1 - base }
7806 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7807 \int_step_inline:nn \l_@@_split_int
7808 {
7809     \group_begin:
7810     \dim_set:Nn \col@sep
7811     { \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
7812     \pgftransformshift
7813     {
7814         \pgfpoint
7815         {
7816             \str_case:on \l_@@_hpos_block_str
7817             {
7818                 l { \l_tmpa_dim + ##1 \l_tmpb_dim - \l_tmpb_dim + \col@sep }
7819                 c { \l_tmpa_dim + ##1 \l_tmpb_dim - 0.5 \l_tmpb_dim }
7820                 r { \l_tmpa_dim + ##1 \l_tmpb_dim - \col@sep }
7821             }
7822         }
7823         { \l_@@_tmpc_dim }
7824     }
7825     \pgfset
7826     {
7827         inner~xsep = \c_zero_dim ,
7828         inner~ysep = \c_zero_dim
7829     }
7830     \pgfnode
7831     { rectangle }
7832     {
7833         \str_case:on \l_@@_hpos_block_str
7834         {
7835             c { base }
7836             l { base~west }
7837             r { base~east }
7838         }
7839     }
7840     { \seq_item:Nn \l_tmpa_seq { ##1 } } { } { } { }
7841     \group_end:
7842 }
7843 \endpgfpicture
7844 }

```

Now the case where there is no ampersand & in the content of the block.

```

7845 {
7846     \bool_if:NTF \l_@@_p_block_bool
7847     {

```

When the final user has used the key p, we have to compute the width.

```

7848         \pgfpicture
7849         \pgfrememberpicturepositiononpagetrue
7850         \pgf@relevantforpicturesizefalse
7851         \bool_if:NTF \l_@@_hpos_of_block_cap_bool
7852         {

```

```

7853         \@@_qpoint:n { col - #2 }
7854         \dim_gset_eq:NN \g_tmpa_dim \pgf@x
7855         \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7856     }
7857     {
7858         \pgfpointanchor { \@@_env: - #1 - #2 - block - short } { west }
7859         \dim_gset_eq:NN \g_tmpa_dim \pgf@x
7860         \pgfpointanchor { \@@_env: - #1 - #2 - block - short } { east }
7861     }
7862     \dim_gset:Nn \g_tmpb_dim { \pgf@x - \g_tmpa_dim }
7863     \endpgfpicture
7864     \hbox_set:Nn \l_@@_cell_box
7865     {
7866         \begin { minipage } [ \str_lowercase:o \l_@@_vpos_block_str ]
7867         { \g_tmpb_dim }
7868         \str_case:on \l_@@_hpos_block_str
7869         { c \centering r \raggedleft l \raggedright j { } }
7870         #6
7871         \end { minipage }
7872     }
7873 }
7874 { \hbox_set:Nn \l_@@_cell_box { \set@color #6 } }
7875 \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.

```

7876 \pgfpicture
7877 \pgfrememberpicturepositiononpagetrue
7878 \pgf@relevantforpicturesizefalse
7879 \bool_lazy_any:nTF
7880 {
7881     { \str_if_empty_p:N \l_@@_vpos_block_str } % added 2024/06/29
7882     { \str_if_eq_p:on \l_@@_vpos_block_str { c } }
7883     { \str_if_eq_p:on \l_@@_vpos_block_str { T } }
7884     { \str_if_eq_p:on \l_@@_vpos_block_str { B } }
7885 }
7886 {

```

If we are in the first column, we must put the block as if it was with the key `r`.

```

7887     \int_if_zero:nT { #2 } { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_r_str }

```

If we are in the last column, we must put the block as if it was with the key `l`.

```

7888     \bool_if:nT \g_@@_last_col_found_bool
7889     {
7890         \int_compare:nNnT { #2 } = \g_@@_col_total_int
7891         { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_l_str }
7892     }

```

`\l_tmpa_tl` will contain the anchor of the PGF node which will be used.

```

7893     \tl_set:Ne \l_tmpa_tl
7894     {
7895         \str_case:on \l_@@_vpos_block_str
7896         {

```

We recall that `\l_@@_vpos_block_str` is empty when the user has not used a key for the vertical position of the block.

```

7897         { } { % added 2024-06-29
7898             \str_case:on \l_@@_hpos_block_str
7899             {
7900                 c { center }
7901                 l { west }
7902                 r { east }
7903                 j { center }

```

```

7904         }
7905     }
7906     c {
7907         \str_case:on \l_@@_hpos_block_str
7908         {
7909             c { center }
7910             l { west }
7911             r { east }
7912             j { center }
7913         }
7914
7915     }
7916     T {
7917         \str_case:on \l_@@_hpos_block_str
7918         {
7919             c { north }
7920             l { north-west }
7921             r { north-east }
7922             j { north }
7923         }
7924
7925     }
7926     B {
7927         \str_case:on \l_@@_hpos_block_str
7928         {
7929             c { south }
7930             l { south-west }
7931             r { south-east }
7932             j { south }
7933         }
7934
7935     }
7936 }
7937
7938 \pgftransformshift
7939 {
7940     \pgfpointanchor
7941     {
7942         \@@_env: - #1 - #2 - block
7943         \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
7944     }
7945     { \l_tmpa_tl }
7946 }
7947 \pgfset
7948 {
7949     inner~xsep = \c_zero_dim ,
7950     inner~ysep = \c_zero_dim
7951 }
7952 \pgfnode
7953 { rectangle }
7954 { \l_tmpa_tl }
7955 { \box_use_drop:N \l_@@_cell_box } { } { }
7956 }

```

End of the case when \l_@@_vpos_block_str is equal to c, T or B. Now, the other cases.

```

7957 {
7958     \pgfextracty \l_tmpa_dim
7959     {
7960         \@@_qpoint:n
7961         {
7962             row - \str_if_eq:onTF \l_@@_vpos_block_str { b } { #3 } { #1 }
7963             - base
7964         }

```

```

7965         }
7966         \dim_sub:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }

```

We retrieve (in `\pgf@x`) the x -value of the center of the block.

```

7967         \pgfpointanchor
7968         {
7969             \@@_env: - #1 - #2 - block
7970             \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
7971         }
7972         {
7973             \str_case:on \l_@@_hpos_block_str
7974             {
7975                 c { center }
7976                 l { west }
7977                 r { east }
7978                 j { center }
7979             }
7980         }

```

We put the label of the block which has been composed in `\l_@@_cell_box`.

```

7981         \pgftransformshift { \pgfpoint \pgf@x \l_tmpa_dim }
7982         \pgfset { inner~sep = \c_zero_dim }
7983         \pgfnode
7984         { rectangle }
7985         {
7986             \str_case:on \l_@@_hpos_block_str
7987             {
7988                 c { base }
7989                 l { base~west }
7990                 r { base~east }
7991                 j { base }
7992             }
7993         }
7994         { \box_use_drop:N \l_@@_cell_box } { } { }
7995     }
7996     \endpgfpicture
7997 }
7998 \group_end:
7999 }

```

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).

```

8000 \cs_new_protected:Npn \@@_stroke_block:nnn #1 #2 #3
8001 {
8002     \group_begin:
8003     \tl_clear:N \l_@@_draw_tl
8004     \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8005     \keys_set_known:nn { nicematrix / BlockStroke } { #1 }
8006     \pgfpicture
8007     \pgfrememberpicturepositiononpagetrue
8008     \pgf@relevantforpicturesizefalse
8009     \tl_if_empty:NF \l_@@_draw_tl
8010     {

```

If the user has used the key `color` of the command `\Block` without value, the color fixed by `\arrayrulecolor` is used.

```

8011         \tl_if_eq:NNTF \l_@@_draw_tl \c_@@_default_tl
8012         { \CT@arc@ }
8013         { \@@_color:o \l_@@_draw_tl }
8014     }
8015     \pgfsetcornersarced
8016     {

```



```

8017 \pgfpoint
8018 { \l_@@_rounded_corners_dim }
8019 { \l_@@_rounded_corners_dim }
8020 }
8021 \@@_cut_on_hyphen:w #2 \q_stop
8022 \int_compare:nNnF \l_tmpa_tl > \c@iRow
8023 {
8024 \int_compare:nNnF \l_tmpb_tl > \c@jCol
8025 {
8026 \@@_qpoint:n { row - \l_tmpa_tl }
8027 \dim_set_eq:NN \l_tmpb_dim \pgf@y
8028 \@@_qpoint:n { col - \l_tmpb_tl }
8029 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
8030 \@@_cut_on_hyphen:w #3 \q_stop
8031 \int_compare:nNnT \l_tmpa_tl > \c@iRow
8032 { \tl_set:No \l_tmpa_tl { \int_use:N \c@iRow } }
8033 \int_compare:nNnT \l_tmpb_tl > \c@jCol
8034 { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
8035 \@@_qpoint:n { row - \int_eval:n { \l_tmpa_tl + 1 } }
8036 \dim_set_eq:NN \l_tmpa_dim \pgf@y
8037 \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
8038 \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
8039 \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
8040 \pgfpathrectanglecorners
8041 { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
8042 { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
8043 \dim_compare:nNnTF \l_@@_rounded_corners_dim = \c_zero_dim
8044 { \pgfusepathqstroke }
8045 { \pgfusepath { stroke } }
8046 }
8047 }
8048 \endpgfpicture
8049 \group_end:
8050 }

```

Here is the set of keys for the command `\@@_stroke_block:nnn`.

```

8051 \keys_define:nn { nicematrix / BlockStroke }
8052 {
8053 color .tl_set:N = \l_@@_draw_tl ,
8054 draw .code:n =
8055 \tl_if_empty:eF { #1 } { \tl_set:Nn \l_@@_draw_tl { #1 } } ,
8056 draw .default:n = default ,
8057 line-width .dim_set:N = \l_@@_line_width_dim ,
8058 rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
8059 rounded-corners .default:n = 4 pt
8060 }

```

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).

```

8061 \cs_new_protected:Npn \@@_vlines_block:nnn #1 #2 #3
8062 {
8063 \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8064 \keys_set_known:nn { nicematrix / BlockBorders } { #1 }
8065 \@@_cut_on_hyphen:w #2 \q_stop
8066 \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
8067 \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
8068 \@@_cut_on_hyphen:w #3 \q_stop
8069 \tl_set:Ne \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
8070 \tl_set:Ne \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
8071 \int_step_inline:nnn \l_@@_tmpd_tl \l_tmpb_tl
8072 {
8073 \use:e

```

```

8074     {
8075         \@@_vline:n
8076         {
8077             position = ##1 ,
8078             start = \l_@@_tmpc_tl ,
8079             end = \int_eval:n { \l_tmpa_tl - 1 } ,
8080             total-width = \dim_use:N \l_@@_line_width_dim
8081         }
8082     }
8083 }
8084 }
8085 \cs_new_protected:Npn \@@_hlines_block:nnn #1 #2 #3
8086 {
8087     \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8088     \keys_set_known:nn { nicematrix / BlockBorders } { #1 }
8089     \@@_cut_on_hyphen:w #2 \q_stop
8090     \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
8091     \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
8092     \@@_cut_on_hyphen:w #3 \q_stop
8093     \tl_set:Ne \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
8094     \tl_set:Ne \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
8095     \int_step_inline:nnn \l_@@_tmpc_tl \l_tmpa_tl
8096     {
8097         \use:e
8098         {
8099             \@@_hline:n
8100             {
8101                 position = ##1 ,
8102                 start = \l_@@_tmpd_tl ,
8103                 end = \int_eval:n { \l_tmpb_tl - 1 } ,
8104                 total-width = \dim_use:N \l_@@_line_width_dim
8105             }
8106         }
8107     }
8108 }

```

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).

```

8109 \cs_new_protected:Npn \@@_stroke_borders_block:nnn #1 #2 #3
8110 {
8111     \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8112     \keys_set_known:nn { nicematrix / BlockBorders } { #1 }
8113     \dim_compare:nNnTF \l_@@_rounded_corners_dim > \c_zero_dim
8114     { \@@_error:n { borders~forbidden } }
8115     {
8116         \tl_clear_new:N \l_@@_borders_tikz_tl
8117         \keys_set:no
8118         { nicematrix / OnlyForTikzInBorders }
8119         \l_@@_borders_clist
8120         \@@_cut_on_hyphen:w #2 \q_stop
8121         \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
8122         \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
8123         \@@_cut_on_hyphen:w #3 \q_stop
8124         \tl_set:Ne \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
8125         \tl_set:Ne \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
8126         \@@_stroke_borders_block_i:
8127     }
8128 }
8129 \hook_gput_code:nnn { begindocument } { . }
8130 {
8131     \cs_new_protected:Npx \@@_stroke_borders_block_i:

```

```

8132     {
8133         \c_@@_pgfortikzpicture_tl
8134         \@@_stroke_borders_block_ii:
8135         \c_@@_endpgfortikzpicture_tl
8136     }
8137 }

8138 \cs_new_protected:Npn \@@_stroke_borders_block_ii:
8139 {
8140     \pgfrememberpicturepositiononpagetrue
8141     \pgf@relevantforpicturesizefalse
8142     \CT@arc@
8143     \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
8144     \clist_if_in:NnT \l_@@_borders_clist { right }
8145     { \@@_stroke_vertical:n \l_tmpb_tl }
8146     \clist_if_in:NnT \l_@@_borders_clist { left }
8147     { \@@_stroke_vertical:n \l_@@_tmpd_tl }
8148     \clist_if_in:NnT \l_@@_borders_clist { bottom }
8149     { \@@_stroke_horizontal:n \l_tmpa_tl }
8150     \clist_if_in:NnT \l_@@_borders_clist { top }
8151     { \@@_stroke_horizontal:n \l_@@_tmpc_tl }
8152 }

8153 \keys_define:nn { nicematrix / OnlyForTikzInBorders }
8154 {
8155     tikz .code:n =
8156         \cs_if_exist:NTF \tikzpicture
8157         { \tl_set:Nn \l_@@_borders_tikz_tl { #1 } }
8158         { \@@_error:n { tikz~in~borders~without~tikz } } ,
8159     tikz .value_required:n = true ,
8160     top .code:n = ,
8161     bottom .code:n = ,
8162     left .code:n = ,
8163     right .code:n = ,
8164     unknown .code:n = \@@_error:n { bad~border }
8165 }

```

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).

```

8166 \cs_new_protected:Npn \@@_stroke_vertical:n #1
8167 {
8168     \@@_qpoint:n \l_@@_tmpc_tl
8169     \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
8170     \@@_qpoint:n \l_tmpa_tl
8171     \dim_set:Nn \l_@@_tmpc_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
8172     \@@_qpoint:n { #1 }
8173     \tl_if_empty:NTF \l_@@_borders_tikz_tl
8174     {
8175         \pgfpathmoveto { \pgfpoint \pgf@x \l_tmpb_dim }
8176         \pgfpathlineto { \pgfpoint \pgf@x \l_@@_tmpc_dim }
8177         \pgfusepathqstroke
8178     }
8179     {
8180         \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
8181         ( \pgf@x , \l_tmpb_dim ) -- ( \pgf@x , \l_@@_tmpc_dim ) ;
8182     }
8183 }

```

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).

```

8184 \cs_new_protected:Npn \@@_stroke_horizontal:n #1
8185 {
8186     \@@_qpoint:n \l_@@_tmpd_tl
8187     \clist_if_in:NnTF \l_@@_borders_clist { left }

```

```

8188 { \dim_set:Nn \l_tmpa_dim { \pgf@x - 0.5 \l_@@_line_width_dim } }
8189 { \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \l_@@_line_width_dim } }
8190 \@@_qpoint:n \l_tmpb_tl
8191 \dim_set:Nn \l_tmpb_dim { \pgf@x + 0.5 \l_@@_line_width_dim }
8192 \@@_qpoint:n { #1 }
8193 \tl_if_empty:NTF \l_@@_borders_tikz_tl
8194 {
8195   \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }
8196   \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
8197   \pgfusepathqstroke
8198 }
8199 {
8200   \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
8201   ( \l_tmpa_dim , \pgf@y ) -- ( \l_tmpb_dim , \pgf@y ) ;
8202 }
8203 }

```

Here is the set of keys for the command `\@@_stroke_borders_block:nnn`.

```

8204 \keys_define:nn { nicematrix / BlockBorders }
8205 {
8206   borders .clist_set:N = \l_@@_borders_clist ,
8207   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
8208   rounded-corners .default:n = 4 pt ,
8209   line-width .dim_set:N = \l_@@_line_width_dim
8210 }

```

The following command will be used if the key `tikz` has been used for the command `\Block`. The arguments `#1` and `#2` are the coordinates of the first cell and `#3` and `#4` the coordinates of the last cell of the block. `#5` is a comma-separated list of the Tikz keys used with the path. However, among those keys, you have added in `nicematrix` a special key `offset` (an offset for the rectangle of the block). That's why we have to extract that key first.

```

8211 \cs_generate_variant:Nn \@@_block_tikz:nnnnn { n n n n o }
8212 \cs_new_protected:Npn \@@_block_tikz:nnnnn #1 #2 #3 #4 #5
8213 {
8214   \begin { tikzpicture }
8215   \@@_clip_with_rounded_corners:
8216   \clist_map_inline:nn { #5 }
8217   {
8218     \keys_set_known:nnN { nicematrix / SpecialOffset } { ##1 } \l_tmpa_tl
8219     \use:e { \exp_not:N \path [ \l_tmpa_tl ] }
8220     (
8221       [
8222         xshift = \dim_use:N \l_@@_offset_dim ,
8223         yshift = - \dim_use:N \l_@@_offset_dim
8224       ]
8225       #1 -| #2
8226     )
8227     rectangle
8228     (
8229       [
8230         xshift = - \dim_use:N \l_@@_offset_dim ,
8231         yshift = \dim_use:N \l_@@_offset_dim
8232       ]
8233       \int_eval:n { #3 + 1 } -| \int_eval:n { #4 + 1 }
8234     ) ;
8235   }
8236   \end { tikzpicture }
8237 }

8238 \keys_define:nn { nicematrix / SpecialOffset }
8239 { 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.

```
8240 \cs_new_protected:Npn \@@_NullBlock:
8241 { \@@_collect_options:n { \@@_NullBlock_i: } }
8242 \NewExpandableDocumentCommand \@@_NullBlock_i: { m m D < > { } +m }
8243 { }
```

28 How to draw the dotted lines transparently

```
8244 \cs_set_protected:Npn \@@_renew_matrix:
8245 {
8246   \RenewDocumentEnvironment { pmatrix } { } {
8247     { \pNiceMatrix }
8248     { \endpNiceMatrix }
8249   \RenewDocumentEnvironment { vmatrix } { } {
8250     { \vNiceMatrix }
8251     { \endvNiceMatrix }
8252   \RenewDocumentEnvironment { Vmatrix } { } {
8253     { \VNiceMatrix }
8254     { \endVNiceMatrix }
8255   \RenewDocumentEnvironment { bmatrix } { } {
8256     { \bNiceMatrix }
8257     { \endbNiceMatrix }
8258   \RenewDocumentEnvironment { Bmatrix } { } {
8259     { \BNiceMatrix }
8260     { \endBNiceMatrix }
8261 }
```

29 Automatic arrays

We will extract some keys and pass the other keys to the environment `{NiceArrayWithDelims}`.

```
8262 \keys_define:nn { nicematrix / Auto }
8263 {
8264   columns-type .tl_set:N = \l_@@_columns_type_tl ,
8265   columns-type .value_required:n = true ,
8266   l .meta:n = { columns-type = l } ,
8267   r .meta:n = { columns-type = r } ,
8268   c .meta:n = { columns-type = c } ,
8269   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8270   delimiters / color .value_required:n = true ,
8271   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
8272   delimiters / max-width .default:n = true ,
8273   delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
8274   delimiters .value_required:n = true ,
8275   rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
8276   rounded-corners .default:n = 4 pt
8277 }
8278 \NewDocumentCommand \AutoNiceMatrixWithDelims
8279 { m m O { } } > { \SplitArgument { 1 } { - } } m O { } m ! O { } }
8280 { \@@_auto_nice_matrix:nnnnn { #1 } { #2 } #4 { #6 } { #3 , #5 , #7 } }
8281 \cs_new_protected:Npn \@@_auto_nice_matrix:nnnnn #1 #2 #3 #4 #5 #6
8282 {
```

The group is for the protection of the keys.

```
8283   \group_begin:
8284   \keys_set_known:nnN { nicematrix / Auto } { #6 } \l_tmpa_tl
```

```

8285 \use:e
8286 {
8287   \exp_not:N \begin { NiceArrayWithDelims } { #1 } { #2 }
8288   { * { #4 } { \exp_not:o \l_@@_columns_type_tl } }
8289   [ \exp_not:o \l_tmpa_tl ]
8290 }
8291 \int_if_zero:nT \l_@@_first_row_int
8292 {
8293   \int_if_zero:nT \l_@@_first_col_int { & }
8294   \prg_replicate:nn { #4 - 1 } { & }
8295   \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
8296 }
8297 \prg_replicate:nn { #3 }
8298 {
8299   \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).

```

8300   \prg_replicate:nn { #4 - 1 } { { } #5 & } #5
8301   \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
8302 }
8303 \int_compare:nNnT \l_@@_last_row_int > { -2 }
8304 {
8305   \int_if_zero:nT \l_@@_first_col_int { & }
8306   \prg_replicate:nn { #4 - 1 } { & }
8307   \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
8308 }
8309 \end { NiceArrayWithDelims }
8310 \group_end:
8311 }
8312 \cs_set_protected:Npn \@@_define_com:nnn #1 #2 #3
8313 {
8314   \cs_set_protected:cpn { #1 AutoNiceMatrix }
8315   {
8316     \bool_gset_true:N \g_@@_delims_bool
8317     \str_gset:Nx \g_@@_name_env_str { #1 AutoNiceMatrix }
8318     \AutoNiceMatrixWithDelims { #2 } { #3 }
8319   }
8320 }
8321 \@@_define_com:nnn p ( )
8322 \@@_define_com:nnn b [ ]
8323 \@@_define_com:nnn v | |
8324 \@@_define_com:nnn V \ | \ |
8325 \@@_define_com:nnn B \{ \}

```

We define also a command \AutoNiceMatrix similar to the environment {NiceMatrix}.

```

8326 \NewDocumentCommand \AutoNiceMatrix { 0 { } m 0 { } m ! 0 { } }
8327 {
8328   \group_begin:
8329   \bool_gset_false:N \g_@@_delims_bool
8330   \AutoNiceMatrixWithDelims . . { #2 } { #4 } [ #1 , #3 , #5 ]
8331   \group_end:
8332 }

```

30 The redefinition of the command \dotfill

```

8333 \cs_set_eq:NN \@@_old_dotfill \dotfill

```

```

8334 \cs_new_protected:Npn \@@_dotfill:
8335 {

```

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}`).

```

8336 \@@_old_dotfill
8337 \tl_gput_right:Nn \g_@@_cell_after_hook_tl \@@_dotfill_i:
8338 }

```

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`.

```

8339 \cs_new_protected:Npn \@@_dotfill_i:
8340 { \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } = \c_zero_dim \@@_old_dotfill }

```

31 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.

```

8341 \cs_new_protected:Npn \@@_diagbox:nn #1 #2
8342 {
8343   \tl_gput_right:Ne \g_@@_pre_code_after_tl
8344   {
8345     \@@_actually_diagbox:nnnnnn
8346     { \int_use:N \c@iRow }
8347     { \int_use:N \c@jCol }
8348     { \int_use:N \c@iRow }
8349     { \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.

```

8350 { \g_@@_row_style_tl \exp_not:n { #1 } }
8351 { \g_@@_row_style_tl \exp_not:n { #2 } }
8352 }

```

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`.

```

8353 \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
8354 {
8355   { \int_use:N \c@iRow }
8356   { \int_use:N \c@jCol }
8357   { \int_use:N \c@iRow }
8358   { \int_use:N \c@jCol }

```

The last argument is for the name of the block.

```

8359 { }
8360 }
8361 }

```

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.

```

8362 \cs_new_protected:Npn \@@_actually_diagbox:nnnnnn #1 #2 #3 #4 #5 #6
8363 {
8364   \pgfpicture
8365   \pgf@relevantforpicturesizefalse
8366   \pgfrememberpicturepositiononpagetrue
8367   \@@_qpoint:n { row - #1 }

```

```

8368 \dim_set_eq:NN \l_tmpa_dim \pgf@y
8369 \@@_qpoint:n { col - #2 }
8370 \dim_set_eq:NN \l_tmpb_dim \pgf@x
8371 \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
8372 \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
8373 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8374 \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
8375 \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
8376 \pgfpathlineto { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
8377 {

```

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.

```

8378 \CT@arc@
8379 \pgfsetroundcap
8380 \pgfusepathqstroke
8381 }
8382 \pgfset { inner~sep = 1 pt }
8383 \pgfscope
8384 \pgftransformshift { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
8385 \pgfnode { rectangle } { south~west }
8386 {
8387 \begin { minipage } { 20 cm }
8388 \@@_math_toggle: #5 \@@_math_toggle:
8389 \end { minipage }
8390 }
8391 { }
8392 { }
8393 \endpgfscope
8394 \pgftransformshift { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
8395 \pgfnode { rectangle } { north~east }
8396 {
8397 \begin { minipage } { 20 cm }
8398 \raggedleft
8399 \@@_math_toggle: #6 \@@_math_toggle:
8400 \end { minipage }
8401 }
8402 { }
8403 { }
8404 \endpgfpicture
8405 }

```

32 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`.

```

8406 \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 `\`.

```

8407 \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`.


```

8408 \cs_new_protected:Npn \@@_CodeAfter_ii:n #1 \end
8409 {
8410   \tl_gput_right:Nn \g_nicematrix_code_after_tl { #1 }
8411   \@@_CodeAfter_iv:n
8412 }

```

We catch the argument of the command `\end` (in `#1`).

```

8413 \cs_new_protected:Npn \@@_CodeAfter_iv:n #1
8414 {

```

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.

```

8415   \str_if_eq:eeTF \currenvir { #1 }
8416   { \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`.

```

8417   {
8418     \tl_gput_right:Nn \g_nicematrix_code_after_tl { \end { #1 } }
8419     \@@_CodeAfter_ii:n
8420   }
8421 }

```

33 The delimiters in the preamble

The command `\@@_delimiter:nnn` will be used to draw delimiters inside the matrix when delimiters are specified in the preamble of the array. It does *not* concern the exterior delimiters added by `{NiceArrayWithDelims}` (and `{pNiceArray}`, `{pNiceMatrix}`, etc.).

A delimiter in the preamble of the array will write an instruction `\@@_delimiter:nnn` in the `\g_@@_pre_code_after_tl` (and also potentially add instructions in the preamble provided to `\array` in order to add space between columns).

The first argument is the type of delimiter (`(`, `[`, `\{`, `)`, `]` or `\}`). The second argument is the number of column. The third argument is a boolean equal to `\c_true_bool` (resp. `\c_false_true`) when the delimiter must be put on the left (resp. right) side.

```

8422 \cs_new_protected:Npn \@@_delimiter:nnn #1 #2 #3
8423 {
8424   \pgfpicture
8425   \pgfrememberpicturepositiononpagetrue
8426   \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.

```

8427   \@@_qpoint:n { row - 1 }
8428   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
8429   \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
8430   \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).

```

8431   \bool_if:nTF { #3 }
8432   { \dim_set_eq:NN \l_tmpa_dim \c_max_dim }
8433   { \dim_set:Nn \l_tmpa_dim { - \c_max_dim } }
8434   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
8435   {
8436     \cs_if_exist:cT
8437     { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
8438     {
8439       \pgfpointanchor

```

```

8440         { \l_@@_env: - ##1 - #2 }
8441         { \bool_if:nTF { #3 } { west } { east } }
8442     \dim_set:Nn \l_tmpa_dim
8443         { \bool_if:nTF { #3 } \dim_min:nn \dim_max:nn \l_tmpa_dim \pgf{x }
8444     }
8445 }

```

Now we can put the delimiter with a node of PGF.

```

8446 \pgfset { inner~sep = \c_zero_dim }
8447 \dim_zero:N \nulldelimiterspace
8448 \pgftransformshift
8449 {
8450     \pgfpoint
8451     { \l_tmpa_dim }
8452     { ( \l_@@_y_initial_dim + \l_@@_y_final_dim + \arrayrulewidth ) / 2 }
8453 }
8454 \pgfnode
8455 { rectangle }
8456 { \bool_if:nTF { #3 } { east } { west } }
8457 {

```

Here is the content of the PGF node, that is to say the delimiter, constructed with its right size.

```

8458     \nullfont
8459     \c_math_toggle_token
8460     \l_@@_color:o \l_@@_delimiters_color_tl
8461     \bool_if:nTF { #3 } { \left #1 } { \left . }
8462     \vcenter
8463     {
8464         \nullfont
8465         \hrule \@height
8466             \dim_eval:n { \l_@@_y_initial_dim - \l_@@_y_final_dim }
8467             \@depth \c_zero_dim
8468             \@width \c_zero_dim
8469     }
8470     \bool_if:nTF { #3 } { \right . } { \right #1 }
8471     \c_math_toggle_token
8472 }
8473 { }
8474 { }
8475 \endpgfpicture
8476 }

```

34 The command `\SubMatrix`

```

8477 \keys_define:nn { nicematrix / sub-matrix }
8478 {
8479     extra-height .dim_set:N = \l_@@_submatrix_extra_height_dim ,
8480     extra-height .value_required:n = true ,
8481     left-xshift .dim_set:N = \l_@@_submatrix_left_xshift_dim ,
8482     left-xshift .value_required:n = true ,
8483     right-xshift .dim_set:N = \l_@@_submatrix_right_xshift_dim ,
8484     right-xshift .value_required:n = true ,
8485     xshift .meta:n = { left-xshift = #1, right-xshift = #1 } ,
8486     xshift .value_required:n = true ,
8487     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8488     delimiters / color .value_required:n = true ,
8489     slim .bool_set:N = \l_@@_submatrix_slim_bool ,
8490     slim .default:n = true ,
8491     hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
8492     hlines .default:n = all ,
8493     vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,

```

```

8494     vlines .default:n = all ,
8495     hvlines .meta:n = { hlines, vlines } ,
8496     hvlines .value_forbidden:n = true
8497   }
8498   \keys_define:nn { nicematrix }
8499   {
8500     SubMatrix .inherit:n = nicematrix / sub-matrix ,
8501     NiceArray / sub-matrix .inherit:n = nicematrix / sub-matrix ,
8502     pNiceArray / sub-matrix .inherit:n = nicematrix / sub-matrix ,
8503     NiceMatrixOptions / sub-matrix .inherit:n = nicematrix / sub-matrix ,
8504   }

```

The following keys set is for the command `\SubMatrix` itself (not the tuning of `\SubMatrix` that can be done elsewhere).

```

8505   \keys_define:nn { nicematrix / SubMatrix }
8506   {
8507     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8508     delimiters / color .value_required:n = true ,
8509     hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
8510     hlines .default:n = all ,
8511     vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
8512     vlines .default:n = all ,
8513     hvlines .meta:n = { hlines, vlines } ,
8514     hvlines .value_forbidden:n = true ,
8515     name .code:n =
8516       \tl_if_empty:nTF { #1 }
8517       { \@@_error:n { Invalid-name } }
8518       {
8519         \regex_match:nnTF { \A[A-Za-z][A-Za-z0-9]*\Z } { #1 }
8520         {
8521           \seq_if_in:NnTF \g_@@_submatrix_names_seq { #1 }
8522           { \@@_error:nn { Duplicate-name-for-SubMatrix } { #1 } }
8523           {
8524             \str_set:Nn \l_@@_submatrix_name_str { #1 }
8525             \seq_gput_right:Nn \g_@@_submatrix_names_seq { #1 }
8526           }
8527         }
8528         { \@@_error:n { Invalid-name } }
8529       } ,
8530     name .value_required:n = true ,
8531     rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
8532     rules .value_required:n = true ,
8533     code .tl_set:N = \l_@@_code_tl ,
8534     code .value_required:n = true ,
8535     unknown .code:n = \@@_error:n { Unknown-key-for-SubMatrix }
8536   }

8537   \NewDocumentCommand \@@_SubMatrix_in_code_before { m m m m ! O { } }
8538   {
8539     \peek_remove_spaces:n
8540     {
8541       \tl_gput_right:Ne \g_@@_pre_code_after_tl
8542       {
8543         \SubMatrix { #1 } { #2 } { #3 } { #4 }
8544         [
8545           delimiters / color = \l_@@_delimiters_color_tl ,
8546           hlines = \l_@@_submatrix_hlines_clist ,
8547           vlines = \l_@@_submatrix_vlines_clist ,
8548           extra-height = \dim_use:N \l_@@_submatrix_extra_height_dim ,
8549           left-xshift = \dim_use:N \l_@@_submatrix_left_xshift_dim ,
8550           right-xshift = \dim_use:N \l_@@_submatrix_right_xshift_dim ,
8551           slim = \bool_to_str:N \l_@@_submatrix_slim_bool ,
8552           #5

```

```

8553     ]
8554   }
8555   \@@_SubMatrix_in_code_before_i { #2 } { #3 }
8556 }
8557 }
8558 \NewDocumentCommand \@@_SubMatrix_in_code_before_i
8559 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
8560 { \@@_SubMatrix_in_code_before_i:nnnn #1 #2 }
8561 \cs_new_protected:Npn \@@_SubMatrix_in_code_before_i:nnnn #1 #2 #3 #4
8562 {
8563   \seq_gput_right:Ne \g_@@_submatrix_seq
8564   {

```

We use `\str_if_eq:nnTF` because it is fully expandable.

```

8565   { \str_if_eq:nnTF { #1 } { last } { \int_use:N \c@iRow } { #1 } }
8566   { \str_if_eq:nnTF { #2 } { last } { \int_use:N \c@jCol } { #2 } }
8567   { \str_if_eq:nnTF { #3 } { last } { \int_use:N \c@iRow } { #3 } }
8568   { \str_if_eq:nnTF { #4 } { last } { \int_use:N \c@jCol } { #4 } }
8569 }
8570 }

```

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`.

```

8571 \hook_gput_code:nnn { begindocument } { . }
8572 {
8573   \cs_set_nopar:Npn \l_@@_argspec_tl { m m m m 0 { } E { _ ^ } { { } { } } }
8574   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
8575   \exp_args:NNo \NewDocumentCommand \@@_SubMatrix \l_@@_argspec_tl
8576   {
8577     \peek_remove_spaces:n
8578     {
8579       \@@_sub_matrix:nnnnnnn
8580       { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } { #7 }
8581     }
8582   }
8583 }

```

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).

```

8584 \NewDocumentCommand \@@_compute_i_j:nn
8585 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
8586 { \@@_compute_i_j:nnnn #1 #2 }

```

```

8587 \cs_new_protected:Npn \@@_compute_i_j:nnnn #1 #2 #3 #4
8588 {
8589   \cs_set_nopar:Npn \l_@@_first_i_tl { #1 }
8590   \cs_set_nopar:Npn \l_@@_first_j_tl { #2 }
8591   \cs_set_nopar:Npn \l_@@_last_i_tl { #3 }
8592   \cs_set_nopar:Npn \l_@@_last_j_tl { #4 }
8593   \tl_if_eq:NnT \l_@@_first_i_tl { last }
8594     { \tl_set:NV \l_@@_first_i_tl \c@iRow }
8595   \tl_if_eq:NnT \l_@@_first_j_tl { last }
8596     { \tl_set:NV \l_@@_first_j_tl \c@jCol }
8597   \tl_if_eq:NnT \l_@@_last_i_tl { last }
8598     { \tl_set:NV \l_@@_last_i_tl \c@iRow }
8599   \tl_if_eq:NnT \l_@@_last_j_tl { last }
8600     { \tl_set:NV \l_@@_last_j_tl \c@jCol }
8601 }
8602 \cs_new_protected:Npn \@@_sub_matrix:nnnnnnn #1 #2 #3 #4 #5 #6 #7
8603 {
8604   \group_begin:

```

The four following token lists correspond to the position of the \SubMatrix.

```

8605   \@@_compute_i_j:nn { #2 } { #3 }
8606   \int_compare:nNnT \l_@@_first_i_tl = \l_@@_last_i_tl
8607     { \cs_set_nopar:Npn \arraystretch { 1 } }
8608   \bool_lazy_or:nnTF
8609     { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
8610     { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
8611     { \@@_error:nn { Construct-too-large } { \SubMatrix } }
8612   {
8613     \str_clear_new:N \l_@@_submatrix_name_str
8614     \keys_set:nn { nicematrix / SubMatrix } { #5 }
8615     \pgfpicture
8616     \pgfrememberpicturepositiononpagetrue
8617     \pgf@relevantforpicturesizefalse
8618     \pgfset { inner~sep = \c_zero_dim }
8619     \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
8620     \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }

```

The last value of \int_step_inline:nnn is provided by currification.

```

8621   \bool_if:NTF \l_@@_submatrix_slim_bool
8622     { \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl }
8623     { \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int }
8624     {
8625       \cs_if_exist:cT
8626         { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
8627       {
8628         \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
8629         \dim_set:Nn \l_@@_x_initial_dim
8630           { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
8631       }
8632       \cs_if_exist:cT
8633         { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
8634       {
8635         \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
8636         \dim_set:Nn \l_@@_x_final_dim
8637           { \dim_max:nn \l_@@_x_final_dim \pgf@x }
8638       }
8639     }
8640   \dim_compare:nNnTF \l_@@_x_initial_dim = \c_max_dim
8641     { \@@_error:nn { Impossible~delimiter } { left } }
8642   {
8643     \dim_compare:nNnTF \l_@@_x_final_dim = { - \c_max_dim }
8644       { \@@_error:nn { Impossible~delimiter } { right } }
8645       { \@@_sub_matrix_i:nnnn { #1 } { #4 } { #6 } { #7 } }
8646   }

```

```

8647     \endpgfpicture
8648   }
8649 \group_end:
8650 }

```

#1 is the left delimiter, #2 is the right one, #3 is the subscript and #4 is the superscript.

```

8651 \cs_new_protected:Npn \@@_sub_matrix_i:nnnn #1 #2 #3 #4
8652 {
8653   \@@_qpoint:n { row - \l_@@_first_i_tl - base }
8654   \dim_set:Nn \l_@@_y_initial_dim
8655   {
8656     \fp_to_dim:n
8657     {
8658       \pgf@y
8659       + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
8660     }
8661   }
8662   \@@_qpoint:n { row - \l_@@_last_i_tl - base }
8663   \dim_set:Nn \l_@@_y_final_dim
8664   { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
8665   \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
8666   {
8667     \cs_if_exist:cT
8668     { pgf @ sh @ ns @ \@@_env: - \l_@@_first_i_tl - ##1 }
8669     {
8670       \pgfpointanchor { \@@_env: - \l_@@_first_i_tl - ##1 } { north }
8671       \dim_set:Nn \l_@@_y_initial_dim
8672       { \dim_max:nn \l_@@_y_initial_dim \pgf@y }
8673     }
8674     \cs_if_exist:cT
8675     { pgf @ sh @ ns @ \@@_env: - \l_@@_last_i_tl - ##1 }
8676     {
8677       \pgfpointanchor { \@@_env: - \l_@@_last_i_tl - ##1 } { south }
8678       \dim_set:Nn \l_@@_y_final_dim
8679       { \dim_min:nn \l_@@_y_final_dim \pgf@y }
8680     }
8681   }
8682   \dim_set:Nn \l_tmpa_dim
8683   {
8684     \l_@@_y_initial_dim - \l_@@_y_final_dim +
8685     \l_@@_submatrix_extra_height_dim - \arrayrulewidth
8686   }
8687   \dim_zero:N \nulldelimiterspace

```

We will draw the rules in the \SubMatrix.

```

8688   \group_begin:
8689   \pgfsetlinewidth { 1.1 \arrayrulewidth }
8690   \@@_set_CT@arc@:o \l_@@_rules_color_tl
8691   \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`.

```

8692   \seq_map_inline:Nn \g_@@_cols_vlism_seq
8693   {
8694     \int_compare:nNnT \l_@@_first_j_tl < { ##1 }
8695     {
8696       \int_compare:nNnT
8697       { ##1 } < { \int_eval:n { \l_@@_last_j_tl + 1 } }
8698       {

```

First, we extract the value of the abscissa of the rule we have to draw.

```

8699     \@@_qpoint:n { col - ##1 }

```

```

8700         \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
8701         \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
8702         \pgfusepathqstroke
8703     }
8704 }
8705 }

```

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.

```

8706 \tl_if_eq:NNTF \l_@@_submatrix_vlines_clist \c_@@_all_tl
8707 { \int_step_inline:nn { \l_@@_last_j_tl - \l_@@_first_j_tl } }
8708 { \clist_map_inline:Nn \l_@@_submatrix_vlines_clist }
8709 {
8710     \bool_lazy_and:nnTF
8711     { \int_compare_p:nNn { ##1 } > \c_zero_int }
8712     {
8713         \int_compare_p:nNn
8714         { ##1 } < { \l_@@_last_j_tl - \l_@@_first_j_tl + 1 } }
8715     {
8716         \@@_qpoint:n { col - \int_eval:n { ##1 + \l_@@_first_j_tl } }
8717         \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
8718         \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
8719         \pgfusepathqstroke
8720     }
8721     { \@@_error:nnn { Wrong~line~in~SubMatrix } { vertical } { ##1 } }
8722 }

```

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.

```

8723 \tl_if_eq:NNTF \l_@@_submatrix_hlines_clist \c_@@_all_tl
8724 { \int_step_inline:nn { \l_@@_last_i_tl - \l_@@_first_i_tl } }
8725 { \clist_map_inline:Nn \l_@@_submatrix_hlines_clist }
8726 {
8727     \bool_lazy_and:nnTF
8728     { \int_compare_p:nNn { ##1 } > \c_zero_int }
8729     {
8730         \int_compare_p:nNn
8731         { ##1 } < { \l_@@_last_i_tl - \l_@@_first_i_tl + 1 } }
8732     {
8733         \@@_qpoint:n { row - \int_eval:n { ##1 + \l_@@_first_i_tl } }

```

We use a group to protect `\l_tmpa_dim` and `\l_tmpb_dim`.

```

8734 \group_begin:

```

We compute in `\l_tmpa_dim` the x -value of the left end of the rule.

```

8735 \dim_set:Nn \l_tmpa_dim
8736 { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
8737 \str_case:nn { #1 }
8738 {
8739     ( { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
8740     [ { \dim_sub:Nn \l_tmpa_dim { 0.2 mm } }
8741     \{ { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
8742 }
8743 \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }

```

We compute in `\l_tmpb_dim` the x -value of the right end of the rule.

```

8744 \dim_set:Nn \l_tmpb_dim
8745 { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
8746 \str_case:nn { #2 }
8747 {
8748     ) { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
8749     ] { \dim_add:Nn \l_tmpb_dim { 0.2 mm } }
8750     \} { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
8751 }

```

```

8752         \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
8753         \pgfusepathqstroke
8754     \group_end:
8755 }
8756 { \@@_error:nnn { Wrong~line~in~SubMatrix } { horizontal } { #1 } }
8757 }

```

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).

```

8758 \str_if_empty:NF \l_@@_submatrix_name_str
8759 {
8760     \@@_pgf_rect_node:nnnnn \l_@@_submatrix_name_str
8761     \l_@@_x_initial_dim \l_@@_y_initial_dim
8762     \l_@@_x_final_dim \l_@@_y_final_dim
8763 }
8764 \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`.

```

8765 \begin { pgfscope }
8766 \pgftransformshift
8767 {
8768     \pgfpoint
8769     { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
8770     { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
8771 }
8772 \str_if_empty:NTF \l_@@_submatrix_name_str
8773 { \@@_node_left:nn #1 { } }
8774 { \@@_node_left:nn #1 { \@@_env: - \l_@@_submatrix_name_str - left } }
8775 \end { pgfscope }

```

Now, we deal with the right delimiter.

```

8776 \pgftransformshift
8777 {
8778     \pgfpoint
8779     { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
8780     { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
8781 }
8782 \str_if_empty:NTF \l_@@_submatrix_name_str
8783 { \@@_node_right:nnnn #2 { } { #3 } { #4 } }
8784 {
8785     \@@_node_right:nnnn #2
8786     { \@@_env: - \l_@@_submatrix_name_str - right } { #3 } { #4 }
8787 }
8788 \cs_set_eq:NN \pgfpointanchor \@@_pgfpointanchor:n
8789 \flag_clear_new:N \l_@@_code_flag
8790 \l_@@_code_tl
8791 }

```

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`.

```

8792 \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.


```

8793 \cs_new_protected:Npn \@@_pgfpointanchor:n #1
8794 {
8795   \use:e
8796   { \exp_not:N \@@_old_pgfpointanchor { \@@_pgfpointanchor_i:nn #1 } }
8797 }

```

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 -.

```

8798 \cs_new:Npn \@@_pgfpointanchor_i:nn #1 #2
8799 { #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.

```

8800 \tl_const:Nn \c_@@_integers_alist_tl
8801 {
8802   { 1 } { } { 2 } { } { 3 } { } { 4 } { } { 5 } { }
8803   { 6 } { } { 7 } { } { 8 } { } { 9 } { } { 10 } { }
8804   { 11 } { } { 12 } { } { 13 } { } { 14 } { } { 15 } { }
8805   { 16 } { } { 17 } { } { 18 } { } { 19 } { } { 20 } { }
8806 }

8807 \cs_new:Npn \@@_pgfpointanchor_ii:w #1-#2\q_stop
8808 {

```

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`.

```

8809   \tl_if_empty:nTF { #2 }
8810   {
8811     \str_case:nVTF { #1 } \c_@@_integers_alist_tl
8812     {
8813       \flag_raise:N \l_@@_code_flag
8814       \int_if_even:nTF { \flag_height:N \l_@@_code_flag }
8815       { \int_eval:n { #1 + \l_@@_first_i_tl - 1 } }
8816       { \int_eval:n { #1 + \l_@@_first_j_tl - 1 } }
8817     }
8818     { #1 }
8819   }

```

If there is an hyphen, we have to see whether we have a node of the form $i-j$, row- i or col- j .

```

8820     { \@@_pgfpointanchor_iii:w { #1 } #2 }
8821   }

```

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`).

```

8822 \cs_new:Npn \@@_pgfpointanchor_iii:w #1 #2 -
8823 {
8824   \str_case:nnF { #1 }
8825   {
8826     { row } { row - \int_eval:n { #2 + \l_@@_first_i_tl - 1 } }
8827     { col } { col - \int_eval:n { #2 + \l_@@_first_j_tl - 1 } }
8828   }

```

Now the case of a node of the form $i-j$.

```

8829   {
8830     \int_eval:n { #1 + \l_@@_first_i_tl - 1 }
8831     - \int_eval:n { #2 + \l_@@_first_j_tl - 1 }
8832   }
8833 }

```

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`).

```

8834 \cs_new_protected:Npn \@@_node_left:nn #1 #2
8835 {
8836   \pgfnode
8837   { rectangle }
8838   { east }
8839   {
8840     \nullfont
8841     \c_math_toggle_token
8842     \@@_color:o \l_@@_delimiters_color_tl
8843     \left #1
8844     \vcenter
8845     {
8846       \nullfont
8847       \hrule \@height \l_tmpa_dim
8848       \@depth \c_zero_dim
8849       \@width \c_zero_dim
8850     }
8851     \right .
8852     \c_math_toggle_token
8853   }
8854   { #2 }
8855   { }
8856 }

```

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.

```

8857 \cs_new_protected:Npn \@@_node_right:nnn #1 #2 #3 #4
8858 {
8859   \pgfnode
8860   { rectangle }
8861   { west }
8862   {
8863     \nullfont
8864     \c_math_toggle_token
8865     \colorlet { current-color } { . }
8866     \@@_color:o \l_@@_delimiters_color_tl
8867     \left .
8868     \vcenter
8869     {
8870       \nullfont
8871       \hrule \@height \l_tmpa_dim
8872       \@depth \c_zero_dim
8873       \@width \c_zero_dim
8874     }
8875     \right #1
8876     \tl_if_empty:nF { #3 } { _ { \smash { #3 } } }
8877     ^ { \color { current-color } \smash { #4 } }
8878     \c_math_toggle_token
8879   }
8880   { #2 }
8881   { }
8882 }

```

35 Les commandes `\UnderBrace` et `\OverBrace`

The following commands will be linked to `\UnderBrace` and `\OverBrace` in the `\CodeAfter`.

```

8883 \NewDocumentCommand \@@_UnderBrace { 0 { } m m m 0 { } }
8884 {
8885   \peek_remove_spaces:n
8886   { \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { under } }
8887 }

8888 \NewDocumentCommand \@@_OverBrace { 0 { } m m m 0 { } }
8889 {
8890   \peek_remove_spaces:n
8891   { \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { over } }
8892 }

8893 \keys_define:nn { nicematrix / Brace }
8894 {
8895   left-shorten .bool_set:N = \l_@@_brace_left_shorten_bool ,
8896   left-shorten .default:n = true ,
8897   left-shorten .value_forbidden:n = true ,
8898   right-shorten .bool_set:N = \l_@@_brace_right_shorten_bool ,
8899   right-shorten .default:n = true ,
8900   right-shorten .value_forbidden:n = true ,
8901   shorten .meta:n = { left-shorten , right-shorten } ,
8902   shorten .value_forbidden:n = true ,
8903   yshift .dim_set:N = \l_@@_brace_yshift_dim ,
8904   yshift .value_required:n = true ,
8905   yshift .initial:n = \c_zero_dim ,
8906   color .tl_set:N = \l_tmpa_tl ,
8907   color .value_required:n = true ,
8908   unknown .code:n = \@@_error:n { Unknown-key~for~Brace }
8909 }

```

#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`.

```

8910 \cs_new_protected:Npn \@@_brace:nnnnn #1 #2 #3 #4 #5
8911 {
8912   \group_begin:

```

The four following token lists correspond to the position of the sub-matrix to which a brace will be attached.

```

8913 \@@_compute_i_j:nn { #1 } { #2 }
8914 \bool_lazy_or:nnTF
8915 { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
8916 { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
8917 {
8918   \str_if_eq:nnTF { #5 } { under }
8919   { \@@_error:nn { Construct-too-large } { \UnderBrace } }
8920   { \@@_error:nn { Construct-too-large } { \OverBrace } }
8921 }
8922 {
8923   \tl_clear:N \l_tmpa_tl
8924   \keys_set:nn { nicematrix / Brace } { #4 }
8925   \tl_if_empty:NF \l_tmpa_tl { \color { \l_tmpa_tl } }
8926   \pgfpicture
8927   \pgfrememberpicturepositiononpagetrue
8928   \pgf@relevantforpicturesizefalse
8929   \bool_if:NT \l_@@_brace_left_shorten_bool
8930   {
8931     \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
8932     \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
8933     {

```

```

8934         \cs_if_exist:cT
8935         { pgf @ sh @ ns @ \l_@@_env: - ##1 - \l_@@_first_j_tl }
8936         {
8937             \pgfpointanchor { \l_@@_env: - ##1 - \l_@@_first_j_tl } { west }
8938             \dim_set:Nn \l_@@_x_initial_dim
8939             { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
8940         }
8941     }
8942 }
8943 \bool_lazy_or:nnT
8944 { \bool_not_p:n \l_@@_brace_left_shorten_bool }
8945 { \dim_compare_p:nNn \l_@@_x_initial_dim = \c_max_dim }
8946 {
8947     \l_@@_qpoint:n { col - \l_@@_first_j_tl }
8948     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
8949 }
8950 \bool_if:NT \l_@@_brace_right_shorten_bool
8951 {
8952     \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
8953     \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
8954     {
8955         \cs_if_exist:cT
8956         { pgf @ sh @ ns @ \l_@@_env: - ##1 - \l_@@_last_j_tl }
8957         {
8958             \pgfpointanchor { \l_@@_env: - ##1 - \l_@@_last_j_tl } { east }
8959             \dim_set:Nn \l_@@_x_final_dim
8960             { \dim_max:nn \l_@@_x_final_dim \pgf@x }
8961         }
8962     }
8963 }
8964 \bool_lazy_or:nnT
8965 { \bool_not_p:n \l_@@_brace_right_shorten_bool }
8966 { \dim_compare_p:nNn \l_@@_x_final_dim = { - \c_max_dim } }
8967 {
8968     \l_@@_qpoint:n { col - \int_eval:n { \l_@@_last_j_tl + 1 } }
8969     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
8970 }
8971 \pgfset { inner~sep = \c_zero_dim }
8972 \str_if_eq:nnTF { #5 } { under }
8973 { \l_@@_underbrace_i:n { #3 } }
8974 { \l_@@_overbrace_i:n { #3 } }
8975 \endpgfpicture
8976 }
8977 \group_end:
8978 }

```

The argument is the text to put above the brace.

```

8979 \cs_new_protected:Npn \l_@@_overbrace_i:n #1
8980 {
8981     \l_@@_qpoint:n { row - \l_@@_first_i_tl }
8982     \pgftransformshift
8983     {
8984         \pgfpoint
8985         { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
8986         { \pgf@y + \l_@@_brace_yshift_dim - 3 pt }
8987     }
8988     \pgfnode
8989     { rectangle }
8990     { south }
8991     {
8992         \vtop
8993         {
8994             \group_begin:
8995             \everycr { }

```

```

8996         \halign
8997         {
8998             \hfil ## \hfil \crcr
8999             \@@_math_toggle: #1 \@@_math_toggle: \cr
9000             \noalign { \skip_vertical:n { 3 pt } \nointerlineskip }
9001             \c_math_toggle_token
9002             \overbrace
9003             {
9004                 \hbox_to_wd:nn
9005                 { \l_@@_x_final_dim - \l_@@_x_initial_dim }
9006                 { }
9007             }
9008             \c_math_toggle_token
9009             \cr
9010         }
9011     \group_end:
9012 }
9013 }
9014 { }
9015 { }
9016 }

```

The argument is the text to put under the brace.

```

9017 \cs_new_protected:Npn \@@_underbrace_i:n #1
9018 {
9019     \@@_qpoint:n { row - \int_eval:n { \l_@@_last_i_tl + 1 } }
9020     \pgftransformshift
9021     {
9022         \pgfpoint
9023         { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
9024         { \pgf@y - \l_@@_brace_yshift_dim + 3 pt }
9025     }
9026     \pgfnode
9027     { rectangle }
9028     { north }
9029     {
9030         \group_begin:
9031         \everycr { }
9032         \vbox
9033         {
9034             \halign
9035             {
9036                 \hfil ## \hfil \crcr
9037                 \c_math_toggle_token
9038                 \underbrace
9039                 {
9040                     \hbox_to_wd:nn
9041                     { \l_@@_x_final_dim - \l_@@_x_initial_dim }
9042                     { }
9043                 }
9044                 \c_math_toggle_token
9045                 \cr
9046                 \noalign { \skip_vertical:n { 3 pt } \nointerlineskip }
9047                 \@@_math_toggle: #1 \@@_math_toggle: \cr
9048             }
9049         }
9050         \group_end:
9051     }
9052     { }
9053     { }
9054 }

```

36 The command TikzEveryCell

```

9055 \bool_new:N \l_@@_not_empty_bool
9056 \bool_new:N \l_@@_empty_bool
9057
9058 \keys_define:nn { nicematrix / TikzEveryCell }
9059 {
9060     not-empty .code:n =
9061         \bool_lazy_or:nnTF
9062             \l_@@_in_code_after_bool
9063             \g_@@_recreate_cell_nodes_bool
9064             { \bool_set_true:N \l_@@_not_empty_bool }
9065             { \@@_error:n { detection-of-empty-cells } } ,
9066     not-empty .value_forbidden:n = true ,
9067     empty .code:n =
9068         \bool_lazy_or:nnTF
9069             \l_@@_in_code_after_bool
9070             \g_@@_recreate_cell_nodes_bool
9071             { \bool_set_true:N \l_@@_empty_bool }
9072             { \@@_error:n { detection-of-empty-cells } } ,
9073     empty .value_forbidden:n = true ,
9074     unknown .code:n = \@@_error:n { Unknown-key-for-TikzEveryCell }
9075 }
9076
9077
9078 \NewDocumentCommand { \@@_TikzEveryCell } { 0 { } m }
9079 {
9080     \IfPackageLoadedTF { tikz }
9081     {
9082         \group_begin:
9083         \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.

```

9084         \tl_set:Nn \l_tmpa_tl { { #2 } }
9085         \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
9086             { \@@_for_a_block:nnnnn #1 }
9087         \@@_all_the_cells:
9088         \group_end:
9089     }
9090     { \@@_error:n { TikzEveryCell-without-tikz } }
9091 }
9092
9093 \tl_new:N \@@_i_tl
9094 \tl_new:N \@@_j_tl
9095
9096 \cs_new_protected:Nn \@@_all_the_cells:
9097 {
9098     \int_step_variable:nNn { \int_use:c { c@iRow } } \@@_i_tl
9099     {
9100         \int_step_variable:nNn { \int_use:c { c@jCol } } \@@_j_tl
9101         {
9102             \cs_if_exist:cF { cell - \@@_i_tl - \@@_j_tl }
9103             {
9104                 \seq_if_in:NcF \l_@@_corners_cells_seq
9105                     { \@@_i_tl - \@@_j_tl }
9106                 {
9107                     \bool_set_false:N \l_tmpa_bool
9108                     \cs_if_exist:cTF
9109                         { pgf @ sh @ ns @ \@@_env: - \@@_i_tl - \@@_j_tl }
9110                         {
9111                             \bool_if:NF \l_@@_empty_bool
9112                             { \bool_set_true:N \l_tmpa_bool }

```

```

9113     }
9114     {
9115         \bool_if:NF \l_@@_not_empty_bool
9116         { \bool_set_true:N \l_tmpa_bool }
9117     }
9118     \bool_if:NT \l_tmpa_bool
9119     {
9120         \@@_block_tikz:nnnno
9121         \@@_i_tl \@@_j_tl \@@_i_tl \@@_j_tl \l_tmpa_tl
9122     }
9123     }
9124     }
9125     }
9126     }
9127 }
9128
9129 \cs_new_protected:Nn \@@_for_a_block:nnnnn
9130 {
9131     \bool_if:NF \l_@@_empty_bool
9132     {
9133         \@@_block_tikz:nnnno
9134         { #1 } { #2 } { #3 } { #4 } \l_tmpa_tl
9135     }
9136     \@@_mark_cells_of_block:nnnn { #1 } { #2 } { #3 } { #4 }
9137 }
9138
9139 \cs_new_protected:Nn \@@_mark_cells_of_block:nnnn
9140 {
9141     \int_step_inline:nnn { #1 } { #3 }
9142     {
9143         \int_step_inline:nnn { #2 } { #4 }
9144         { \cs_set:cpn { cell - ##1 - ####1 } { } }
9145     }
9146 }

```

37 The command \ShowCellNames

```

9147 \NewDocumentCommand \@@_ShowCellNames_CodeBefore { }
9148 {
9149     \dim_gzero_new:N \g_@@_tmpc_dim
9150     \dim_gzero_new:N \g_@@_tmpd_dim
9151     \dim_gzero_new:N \g_@@_tmpe_dim
9152     \int_step_inline:nn \c@iRow
9153     {
9154         \begin { pgfpicture }
9155         \@@_qpoint:n { row - ##1 }
9156         \dim_set_eq:NN \l_tmpa_dim \pgf@y
9157         \@@_qpoint:n { row - \int_eval:n { ##1 + 1 } }
9158         \dim_gset:Nn \g_tmpa_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
9159         \dim_gset:Nn \g_tmpb_dim { \l_tmpa_dim - \pgf@y }
9160         \bool_if:NTF \l_@@_in_code_after_bool
9161         \end { pgfpicture }
9162         \int_step_inline:nn \c@jCol
9163         {
9164             \hbox_set:Nn \l_tmpa_box
9165             { \normalfont \Large \color { red ! 50 } ##1 - ####1 }
9166             \begin { pgfpicture }
9167             \@@_qpoint:n { col - ####1 }
9168             \dim_gset_eq:NN \g_@@_tmpc_dim \pgf@x
9169             \@@_qpoint:n { col - \int_eval:n { ####1 + 1 } }
9170             \dim_gset:Nn \g_@@_tmpd_dim { \pgf@x - \g_@@_tmpc_dim }
9171             \dim_gset_eq:NN \g_@@_tmpe_dim \pgf@x

```

```

9172 \endpgfpicture
9173 \end { pgfpicture }
9174 \fp_set:Nn \l_tmpa_fp
9175 {
9176     \fp_min:nn
9177     {
9178         \fp_min:nn
9179         {
9180             \dim_ratio:nn
9181             { \g_@@_tmpd_dim }
9182             { \box_wd:N \l_tmpa_box }
9183         }
9184         {
9185             \dim_ratio:nn
9186             { \g_tmpb_dim }
9187             { \box_ht_plus_dp:N \l_tmpa_box }
9188         }
9189     }
9190     { 1.0 }
9191 }
9192 \box_scale:Nnn \l_tmpa_box
9193 { \fp_use:N \l_tmpa_fp }
9194 { \fp_use:N \l_tmpa_fp }
9195 \pgfpicture
9196 \pgfrememberpicturepositiononpagetrue
9197 \pgf@relevantforpicturesizefalse
9198 \pgftransformshift
9199 {
9200     \pgfpoint
9201     { 0.5 * ( \g_@@_tmpc_dim + \g_@@_tmpe_dim ) }
9202     { \dim_use:N \g_tmpa_dim }
9203 }
9204 \pgfnode
9205 { rectangle }
9206 { center }
9207 { \box_use:N \l_tmpa_box }
9208 { }
9209 { }
9210 \endpgfpicture
9211 }
9212 }
9213 }
9214 \NewDocumentCommand \@@_ShowCellNames { }
9215 {
9216     \bool_if:NT \l_@@_in_code_after_bool
9217     {
9218         \pgfpicture
9219         \pgfrememberpicturepositiononpagetrue
9220         \pgf@relevantforpicturesizefalse
9221         \pgfpathrectanglecorners
9222         { \@@_qpoint:n { 1 } }
9223         {
9224             \@@_qpoint:n
9225             { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
9226         }
9227         \pgfsetfillopacity { 0.75 }
9228         \pgfsetfillcolor { white }
9229         \pgfusepathqfill
9230         \endpgfpicture
9231     }
9232     \dim_gzero_new:N \g_@@_tmpc_dim
9233     \dim_gzero_new:N \g_@@_tmpd_dim
9234     \dim_gzero_new:N \g_@@_tmpe_dim

```



```

9235 \int_step_inline:nn \c@iRow
9236 {
9237   \bool_if:NTF \l_@@_in_code_after_bool
9238   {
9239     \pgfpicture
9240     \pgfrememberpicturepositiononpagetrue
9241     \pgf@relevantforpicturesizefalse
9242   }
9243   { \begin { pgfpicture } }
9244   \@@_qpoint:n { row - ##1 }
9245   \dim_set_eq:NN \l_tmpa_dim \pgf@y
9246   \@@_qpoint:n { row - \int_eval:n { ##1 + 1 } }
9247   \dim_gset:Nn \g_tmpa_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
9248   \dim_gset:Nn \g_tmpb_dim { \l_tmpa_dim - \pgf@y }
9249   \bool_if:NTF \l_@@_in_code_after_bool
9250   { \endpgfpicture }
9251   { \end { pgfpicture } }
9252   \int_step_inline:nn \c@jCol
9253   {
9254     \hbox_set:Nn \l_tmpa_box
9255     {
9256       \normalfont \Large \sffamily \bfseries
9257       \bool_if:NTF \l_@@_in_code_after_bool
9258       { \color { red } }
9259       { \color { red ! 50 } }
9260       ##1 - ####1
9261     }
9262     \bool_if:NTF \l_@@_in_code_after_bool
9263     {
9264       \pgfpicture
9265       \pgfrememberpicturepositiononpagetrue
9266       \pgf@relevantforpicturesizefalse
9267     }
9268     { \begin { pgfpicture } }
9269     \@@_qpoint:n { col - ####1 }
9270     \dim_gset_eq:NN \g_@@_tmpc_dim \pgf@x
9271     \@@_qpoint:n { col - \int_eval:n { ####1 + 1 } }
9272     \dim_gset:Nn \g_@@_tmpd_dim { \pgf@x - \g_@@_tmpc_dim }
9273     \dim_gset_eq:NN \g_@@_tmpe_dim \pgf@x
9274     \bool_if:NTF \l_@@_in_code_after_bool
9275     { \endpgfpicture }
9276     { \end { pgfpicture } }
9277     \fp_set:Nn \l_tmpa_fp
9278     {
9279       \fp_min:nn
9280       {
9281         \fp_min:nn
9282         { \dim_ratio:nn \g_@@_tmpd_dim { \box_wd:N \l_tmpa_box } }
9283         { \dim_ratio:nn \g_tmpb_dim { \box_ht_plus_dp:N \l_tmpa_box } }
9284       }
9285       { 1.0 }
9286     }
9287     \box_scale:Nnn \l_tmpa_box { \fp_use:N \l_tmpa_fp } { \fp_use:N \l_tmpa_fp }
9288     \pgfpicture
9289     \pgfrememberpicturepositiononpagetrue
9290     \pgf@relevantforpicturesizefalse
9291     \pgftransformshift
9292     {
9293       \pgfpoint
9294       { 0.5 * ( \g_@@_tmpc_dim + \g_@@_tmpe_dim ) }
9295       { \dim_use:N \g_tmpa_dim }
9296     }
9297     \pgfnode

```

```

9298         { rectangle }
9299         { center }
9300         { \box_use:N \l_tmpa_box }
9301         { }
9302         { }
9303     \endpgfpicture
9304 }
9305 }
9306 }

```

38 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.

```

9307 \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.

```

9308 \bool_new:N \g_@@_footnote_bool
9309 \msg_new:nnnn { nicematrix } { Unknown~key~for~package }
9310 {
9311     The~key~'\l_keys_key_str'~is~unknown. \\
9312     That~key~will~be~ignored. \\
9313     For~a~list~of~the~available~keys,~type~H~<return>.
9314 }
9315 {
9316     The~available~keys~are~(in~alphabetic~order):~
9317     footnote,~
9318     footnotehyper,~
9319     messages-for-Overleaf,~
9320     no-test-for-array,~
9321     renew-dots,~and~
9322     renew-matrix.
9323 }
9324 \keys_define:nn { nicematrix / Package }
9325 {
9326     renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
9327     renew-dots .value_forbidden:n = true ,
9328     renew-matrix .code:n = \@@_renew_matrix: ,
9329     renew-matrix .value_forbidden:n = true ,
9330     messages-for-Overleaf .bool_set:N = \g_@@_messages_for_Overleaf_bool ,
9331     footnote .bool_set:N = \g_@@_footnote_bool ,
9332     footnotehyper .bool_set:N = \g_@@_footnotehyper_bool ,
9333     no-test-for-array .bool_set:N = \g_@@_no_test_for_array_bool ,
9334     no-test-for-array .default:n = true ,
9335     unknown .code:n = \@@_error:n { Unknown~key~for~package }
9336 }
9337 \ProcessKeysOptions { nicematrix / Package }
9338 \@@_msg_new:nn { footnote-with-footnotehyper-package }
9339 {
9340     You~can't~use~the~option~'footnote'~because~the~package~
9341     footnotehyper~has~already~been~loaded.~
9342     If~you~want,~you~can~use~the~option~'footnotehyper'~and~the~footnotes~

```

```

9343     within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
9344     of~the~package~footnotehyper.\\
9345     The~package~footnote~won't~be~loaded.
9346   }
9347   \@@_msg_new:nn { footnotehyper~with~footnote~package }
9348   {
9349     You~can't~use~the~option~'footnotehyper'~because~the~package~
9350     footnote~has~already~been~loaded.~
9351     If~you~want,~you~can~use~the~option~'footnote'~and~the~footnotes~
9352     within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
9353     of~the~package~footnote.\\
9354     The~package~footnotehyper~won't~be~loaded.
9355   }

9356   \bool_if:NT \g_@@_footnote_bool
9357   {

```

The class `beamer` has its own system to extract footnotes and that's why we have nothing to do if `beamer` is used.

```

9358     \IfClassLoadedTF { beamer }
9359     { \bool_set_false:N \g_@@_footnote_bool }
9360     {
9361       \IfPackageLoadedTF { footnotehyper }
9362       { \@@_error:n { footnote~with~footnotehyper~package } }
9363       { \usepackage { footnote } }
9364     }
9365   }

9366   \bool_if:NT \g_@@_footnotehyper_bool
9367   {

```

The class `beamer` has its own system to extract footnotes and that's why we have nothing to do if `beamer` is used.

```

9368     \IfClassLoadedTF { beamer }
9369     { \bool_set_false:N \g_@@_footnote_bool }
9370     {
9371       \IfPackageLoadedTF { footnote }
9372       { \@@_error:n { footnotehyper~with~footnote~package } }
9373       { \usepackage { footnotehyper } }
9374     }
9375     \bool_set_true:N \g_@@_footnote_bool
9376   }

```

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}`.

39 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.

```

9377   \bool_new:N \l_@@_underscore_loaded_bool
9378   \IfPackageLoadedT { underscore }
9379   { \bool_set_true:N \l_@@_underscore_loaded_bool }

9380   \hook_gput_code:nnn { begindocument } { . }
9381   {
9382     \bool_if:NF \l_@@_underscore_loaded_bool
9383     {
9384       \IfPackageLoadedT { underscore }
9385       { \@@_error:n { underscore~after~nicematrix } }
9386     }
9387   }

```

40 Error messages of the package

```

9388 \bool_if:NTF \g_@@_messages_for_Overleaf_bool
9389 { \str_const:Nn \c_@@_available_keys_str { } }
9390 {
9391   \str_const:Nn \c_@@_available_keys_str
9392     { For~a~list~of~the~available~keys,~type~H~<return>. }
9393 }

9394 \seq_new:N \g_@@_types_of_matrix_seq
9395 \seq_gset_from_clist:Nn \g_@@_types_of_matrix_seq
9396 {
9397   NiceMatrix ,
9398   pNiceMatrix , bNiceMatrix , vNiceMatrix, BNiceMatrix, VNiceMatrix
9399 }
9400 \seq_gset_map_x:NNn \g_@@_types_of_matrix_seq \g_@@_types_of_matrix_seq
9401 { \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`.

```

9402 \cs_new_protected:Npn \@@_error_too_much_cols:
9403 {
9404   \seq_if_in:NoF \g_@@_types_of_matrix_seq \g_@@_name_env_str
9405     { \@@_fatal:nn { too-much-cols-for-array } }
9406   \int_compare:nNnT \l_@@_last_col_int = { -2 }
9407     { \@@_fatal:n { too-much-cols-for-matrix } }
9408   \int_compare:nNnT \l_@@_last_col_int = { -1 }
9409     { \@@_fatal:n { too-much-cols-for-matrix } }
9410   \bool_if:NF \l_@@_last_col_without_value_bool
9411     { \@@_fatal:n { too-much-cols-for-matrix-with-last-col } }
9412 }

```

The following command must *not* be protected since it's used in an error message.

```

9413 \cs_new:Npn \@@_message_hdotsfor:
9414 {
9415   \tl_if_empty:oF \g_@@_HVdotsfor_lines_tl
9416     { ~Maybe~your~use~of~\token_to_str:N \Hdotsfor\ is~incorrect.}
9417 }

9418 \@@_msg_new:nn { hvlines,~rounded-corners~and~corners }
9419 {
9420   Incompatible~options.\\
9421   You~should~not~use~'hvlines',~'rounded-corners'~and~'corners'~at~this~time.\\
9422   The~output~will~not~be~reliable.
9423 }

9424 \@@_msg_new:nn { negative~weight }
9425 {
9426   Negative~weight.\\
9427   The~weight~of~the~'X'~columns~must~be~positive~and~you~have~used~
9428   the~value~'\int_use:N \l_@@_weight_int'.\\
9429   The~absolute~value~will~be~used.
9430 }

9431 \@@_msg_new:nn { last-col~not~used }
9432 {
9433   Column~not~used.\\
9434   The~key~'last-col'~is~in~force~but~you~have~not~used~that~last~column~
9435   in~your~\@@_full_name_env:.~However,~you~can~go~on.
9436 }

9437 \@@_msg_new:nn { too-much-cols-for-matrix-with-last-col }
9438 {

```

```

9439 Too-much-columns.\\
9440 In~the~row~\int_eval:n { \c@iRow },~
9441 you~try~to~use~more~columns~
9442 than~allowed~by~your~\@@_full_name_env:.\@@_message_hdotsfor:\
9443 The~maximal~number~of~columns~is~\int_eval:n { \l_@@_last_col_int - 1 }~
9444 (plus~the~exterior~columns).~This~error~is~fatal.
9445 }

9446 \@@_msg_new:nn { too-much-cols-for-matrix }
9447 {
9448   Too-much-columns.\\
9449   In~the~row~\int_eval:n { \c@iRow },~
9450   you~try~to~use~more~columns~than~allowed~by~your~
9451   \@@_full_name_env:.\@@_message_hdotsfor:\ Recall~that~the~maximal~
9452   number~of~columns~for~a~matrix~(excepted~the~potential~exterior~
9453   columns)~is~fixed~by~the~LaTeX~counter~'MaxMatrixCols'.~
9454   Its~current~value~is~\int_use:N \c@MaxMatrixCols\ (use~
9455   \token_to_str:N \setcounter\ to~change~that~value).~
9456   This~error~is~fatal.
9457 }

9458 \@@_msg_new:nn { too-much-cols-for-array }
9459 {
9460   Too-much-columns.\\
9461   In~the~row~\int_eval:n { \c@iRow },~
9462   ~you~try~to~use~more~columns~than~allowed~by~your~
9463   \@@_full_name_env:.\@@_message_hdotsfor:\ The~maximal~number~of~columns~is~
9464   \int_use:N \g_@@_static_num_of_col_int\
9465   ~(plus~the~potential~exterior~ones).~
9466   This~error~is~fatal.
9467 }

9468 \@@_msg_new:nn { columns-not-used }
9469 {
9470   Columns-not-used.\\
9471   The~preamble~of~your~\@@_full_name_env:\ announces~\int_use:N
9472   \g_@@_static_num_of_col_int\ columns~but~you~use~only~\int_use:N \c@jCol.\\
9473   The~columns~you~did~not~used~won't~be~created.\\
9474   You~won't~have~similar~error~message~till~the~end~of~the~document.
9475 }

9476 \@@_msg_new:nn { empty-preamble }
9477 {
9478   Empty-preamble.\\
9479   The~preamble~of~your~\@@_full_name_env:\ is~empty.\\
9480   This~error~is~fatal.
9481 }

9482 \@@_msg_new:nn { in-first-col }
9483 {
9484   Erroneous-use.\\
9485   You~can't~use~the~command~\#1 in~the~first~column~(number~0)~of~the~array.\\
9486   That~command~will~be~ignored.
9487 }

9488 \@@_msg_new:nn { in-last-col }
9489 {
9490   Erroneous-use.\\
9491   You~can't~use~the~command~\#1 in~the~last~column~(exterior)~of~the~array.\\
9492   That~command~will~be~ignored.
9493 }

9494 \@@_msg_new:nn { in-first-row }
9495 {
9496   Erroneous-use.\\
9497   You~can't~use~the~command~\#1 in~the~first~row~(number~0)~of~the~array.\\
9498   That~command~will~be~ignored.

```

```

9499 }
9500 \@@_msg_new:nn { in~last~row }
9501 {
9502     You~can't~use~the~command~\#1~in~the~last~row~(exterior)~of~the~array.\\
9503     That~command~will~be~ignored.
9504 }
9505 \@@_msg_new:nn { caption~outside~float }
9506 {
9507     Key~caption~forbidden.\\
9508     You~can't~use~the~key~'caption'~because~you~are~not~in~a~floating~
9509     environment.~This~key~will~be~ignored.
9510 }
9511 \@@_msg_new:nn { short~caption~without~caption }
9512 {
9513     You~should~not~use~the~key~'short~caption'~without~'caption'.~
9514     However,~your~'short~caption'~will~be~used~as~'caption'.
9515 }
9516 \@@_msg_new:nn { double~closing~delimiter }
9517 {
9518     Double~delimiter.\\
9519     You~can't~put~a~second~closing~delimiter~\#1"~just~after~a~first~closing~
9520     delimiter.~This~delimiter~will~be~ignored.
9521 }
9522 \@@_msg_new:nn { delimiter~after~opening }
9523 {
9524     Double~delimiter.\\
9525     You~can't~put~a~second~delimiter~\#1"~just~after~a~first~opening~
9526     delimiter.~That~delimiter~will~be~ignored.
9527 }
9528 \@@_msg_new:nn { bad~option~for~line~style }
9529 {
9530     Bad~line~style.\\
9531     Since~you~haven't~loaded~Tikz,~the~only~value~you~can~give~to~'line~style'~
9532     is~'standard'.~That~key~will~be~ignored.
9533 }
9534 \@@_msg_new:nn { Identical~notes~in~caption }
9535 {
9536     Identical~tabular~notes.\\
9537     You~can't~put~several~notes~with~the~same~content~in~
9538     \token_to_str:N \caption\ (but~you~can~in~the~main~tabular).\\
9539     If~you~go~on,~the~output~will~probably~be~erroneous.
9540 }
9541 \@@_msg_new:nn { tabularnote~below~the~tabular }
9542 {
9543     \token_to_str:N \tabularnote\ forbidden\\
9544     You~can't~use~\token_to_str:N \tabularnote\ in~the~caption~
9545     of~your~tabular~because~the~caption~will~be~composed~below~
9546     the~tabular.~If~you~want~the~caption~above~the~tabular~use~the~
9547     key~'caption~above'~in~\token_to_str:N \NiceMatrixOptions.\\
9548     Your~\token_to_str:N \tabularnote\ will~be~discarded~and~
9549     no~similar~error~will~raised~in~this~document.
9550 }
9551 \@@_msg_new:nn { Unknown~key~for~rules }
9552 {
9553     Unknown~key.\\
9554     There~is~only~two~keys~available~here:~width~and~color.\\
9555     Your~key~'\l_keys_key_str'~will~be~ignored.
9556 }
9557 \@@_msg_new:nn { Unknown~key~for~TikzEveryCell }

```

```

9558 {
9559   Unknown~key.\\
9560   There~is~only~two~keys~available~here::~
9561   'empty'~and~'not-empty'.\\
9562   Your~key~'\l_keys_key_str'~will~be~ignored.
9563 }
9564 \@@_msg_new:nn { Unknown~key~for~rotate }
9565 {
9566   Unknown~key.\\
9567   The~only~key~available~here~is~'c'.\\
9568   Your~key~'\l_keys_key_str'~will~be~ignored.
9569 }
9570 \@@_msg_new:nnn { Unknown~key~for~custom~line }
9571 {
9572   Unknown~key.\\
9573   The~key~'\l_keys_key_str'~is~unknown~in~a~'custom~line'.~
9574   It~you~go~on,~you~will~probably~have~other~errors. \\
9575   \c_@@_available_keys_str
9576 }
9577 {
9578   The~available~keys~are~(in~alphabetic~order):~
9579   ccommand,~
9580   color,~
9581   command,~
9582   dotted,~
9583   letter,~
9584   multiplicity,~
9585   sep~color,~
9586   tikz,~and~total~width.
9587 }
9588 \@@_msg_new:nnn { Unknown~key~for~xdots }
9589 {
9590   Unknown~key.\\
9591   The~key~'\l_keys_key_str'~is~unknown~for~a~command~for~drawing~dotted~rules.\\
9592   \c_@@_available_keys_str
9593 }
9594 {
9595   The~available~keys~are~(in~alphabetic~order):~
9596   'color',~
9597   'horizontal~labels',~
9598   'inter',~
9599   'line~style',~
9600   'radius',~
9601   'shorten',~
9602   'shorten~end'~and~'shorten~start'.
9603 }
9604 \@@_msg_new:nn { Unknown~key~for~rowcolors }
9605 {
9606   Unknown~key.\\
9607   As~for~now,~there~is~only~two~keys~available~here::~'cols'~and~'respect~blocks'~
9608   (and~you~try~to~use~'\l_keys_key_str')\\
9609   That~key~will~be~ignored.
9610 }
9611 \@@_msg_new:nn { label~without~caption }
9612 {
9613   You~can't~use~the~key~'label'~in~your~'{NiceTabular}'~because~
9614   you~have~not~used~the~key~'caption'.~The~key~'label'~will~be~ignored.
9615 }
9616 \@@_msg_new:nn { W~warning }
9617 {
9618   Line~\msg_line_number:.~The~cell~is~too~wide~for~your~column~'W'~

```

```

9619     (row~\int_use:N \c@iRow).
9620 }
9621 \@@_msg_new:nn { Construct-too-large }
9622 {
9623     Construct-too-large.\
9624     Your-command~\token_to_str:N #1
9625     can't-be-drawn-because-your-matrix-is-too-small.\
9626     That-command-will-be-ignored.
9627 }
9628 \@@_msg_new:nn { underscore-after-nicematrix }
9629 {
9630     Problem-with-'underscore'.\
9631     The-package-'underscore'~should-be-loaded-before-'nicematrix'.~
9632     You-can-go-on-but-you-won't-be-able-to-write-something-such-as:\
9633     '\token_to_str:N \Cdots\token_to_str:N _{n~\token_to_str:N \text{~times}}'.
9634 }
9635 \@@_msg_new:nn { ampersand-in-light-syntax }
9636 {
9637     Ampersand-forbidden.\
9638     You-can't-use-an-ampersand~(\token_to_str:N &)~to-separate-columns-because~
9639     ~the-key-'light-syntax'~is-in-force.~This-error-is-fatal.
9640 }
9641 \@@_msg_new:nn { double-backslash-in-light-syntax }
9642 {
9643     Double-backslash-forbidden.\
9644     You-can't-use~\token_to_str:N
9645     \~to-separate-rows-because-the-key-'light-syntax'~
9646     is-in-force.~You-must-use-the-character~'\l_@@_end_of_row_tl'~
9647     (set-by-the-key-'end-of-row').~This-error-is-fatal.
9648 }
9649 \@@_msg_new:nn { hlines-with-color }
9650 {
9651     Incompatible-keys.\
9652     You-can't-use-the-keys-'hlines',~'vlines'~or~'hvlines'~for-a~
9653     '\token_to_str:N \Block'~when-the-key-'color'~or~'draw'~is-used.\
9654     However,~you-can-put-several-commands~\token_to_str:N \Block.\
9655     Your-key-will-be-discarded.
9656 }
9657 \@@_msg_new:nn { bad-value-for-baseline }
9658 {
9659     Bad-value-for-baseline.\
9660     The-value-given-to-'baseline'~(\int_use:N \l_tmpa_int)~is-not~
9661     valid.~The-value-must-be-between~\int_use:N \l_@@_first_row_int~and~
9662     \int_use:N \g_@@_row_total_int~or-equal-to~'t',~'c'~or~'b'~or-of~
9663     the-form-'line-i'.\
9664     A-value-of~1~will-be-used.
9665 }
9666 \@@_msg_new:nn { detection-of-empty-cells }
9667 {
9668     Problem-with-'not-empty'~\
9669     For-technical-reasons,~you-must-activate~
9670     'create-cell-nodes'~in~\token_to_str:N \CodeBefore\
9671     in-order-to-use-the-key~'\l_keys_key_str'~.\
9672     That-key-will-be-ignored.
9673 }
9674 \@@_msg_new:nn { siunitx-not-loaded }
9675 {
9676     siunitx-not-loaded~\
9677     You-can't-use-the-columns-'S'~because-'siunitx'~is-not-loaded.\
9678     That-error-is-fatal.

```



```

9679     }
9680 \@@_msg_new:nn { ragged2e~not~loaded }
9681 {
9682     You~have~to~load~'ragged2e'~in~order~to~use~the~key~'\l_keys_key_str'~in~
9683     your~column~'\l_@@_vpos_col_str'~(or~'X').~The~key~'\str_lowercase:o
9684     \l_keys_key_str'~will~be~used~instead.
9685 }
9686 \@@_msg_new:nn { Invalid~name }
9687 {
9688     Invalid~name.\\
9689     You~can't~give~the~name~'\l_keys_value_tl'~to~a~\token_to_str:N
9690     \SubMatrix\ of~your~\@@_full_name_env:\\.\\
9691     A~name~must~be~accepted~by~the~regular~expression~[A-Za-z][A-Za-z0-9]*.\\
9692     This~key~will~be~ignored.
9693 }
9694 \@@_msg_new:nn { Wrong~line~in~SubMatrix }
9695 {
9696     Wrong~line.\\
9697     You~try~to~draw~a~#1~line~of~number~'#2'~in~a~
9698     \token_to_str:N \SubMatrix\ of~your~\@@_full_name_env:\ but~that~
9699     number~is~not~valid.~It~will~be~ignored.
9700 }
9701 \@@_msg_new:nn { Impossible~delimiter }
9702 {
9703     Impossible~delimiter.\\
9704     It's~impossible~to~draw~the~#1~delimiter~of~your~
9705     \token_to_str:N \SubMatrix\ because~all~the~cells~are~empty~
9706     in~that~column.
9707     \bool_if:NT \l_@@_submatrix_slim_bool
9708     { ~Maybe~you~should~try~without~the~key~'slim'. } \\
9709     This~\token_to_str:N \SubMatrix\ will~be~ignored.
9710 }
9711 \@@_msg_new:nnn { width~without~X~columns }
9712 {
9713     You~have~used~the~key~'width'~but~you~have~put~no~'X'~column.~
9714     That~key~will~be~ignored.
9715 }
9716 {
9717     This~message~is~the~message~'width~without~X~columns'~
9718     of~the~module~'nicematrix'.~
9719     The~experimented~users~can~disable~that~message~with~
9720     \token_to_str:N \msg_redirect_name:nnn.\\
9721 }
9722
9723 \@@_msg_new:nn { key~multiplicity~with~dotted }
9724 {
9725     Incompatible~keys. \\
9726     You~have~used~the~key~'multiplicity'~with~the~key~'dotted'~
9727     in~a~'custom~line'.~They~are~incompatible. \\
9728     The~key~'multiplicity'~will~be~discarded.
9729 }
9730 \@@_msg_new:nn { empty~environment }
9731 {
9732     Empty~environment.\\
9733     Your~\@@_full_name_env:\ is~empty.~This~error~is~fatal.
9734 }
9735 \@@_msg_new:nn { No~letter~and~no~command }
9736 {
9737     Erroneous~use.\\
9738     Your~use~of~'custom~line'~is~no~op~since~you~don't~have~used~the~

```

```

9739     key~'letter'~(for~a~letter~for~vertical~rules)~nor~the~keys~'command'~or~
9740     ~'ccommand'~(to~draw~horizontal~rules).\\
9741     However,~you~can~go~on.
9742 }

9743 \@@_msg_new:nn { Forbidden~letter }
9744 {
9745     Forbidden~letter.\\
9746     You~can't~use~the~letter~'#1'~for~a~customized~line.\\
9747     It~will~be~ignored.
9748 }

9749 \@@_msg_new:nn { Several~letters }
9750 {
9751     Wrong~name.\\
9752     You~must~use~only~one~letter~as~value~for~the~key~'letter'~(and~you~
9753     have~used~'\l_@@_letter_str').\\
9754     It~will~be~ignored.
9755 }

9756 \@@_msg_new:nn { Delimiter~with~small }
9757 {
9758     Delimiter~forbidden.\\
9759     You~can't~put~a~delimiter~in~the~preamble~of~your~\@@_full_name_env:\\
9760     because~the~key~'small'~is~in~force.\\
9761     This~error~is~fatal.
9762 }

9763 \@@_msg_new:nn { unknown~cell~for~line~in~CodeAfter }
9764 {
9765     Unknown~cell.\\
9766     Your~command~\token_to_str:N\line\{#1\}\{#2\}~in~
9767     the~\token_to_str:N \CodeAfter\ of~your~\@@_full_name_env:\\
9768     can't~be~executed~because~a~cell~doesn't~exist.\\
9769     This~command~\token_to_str:N \line\ will~be~ignored.
9770 }

9771 \@@_msg_new:nnn { Duplicate~name~for~SubMatrix }
9772 {
9773     Duplicate~name.\\
9774     The~name~'#1'~is~already~used~for~a~\token_to_str:N \SubMatrix\\
9775     in~this~\@@_full_name_env:.\\
9776     This~key~will~be~ignored.\\
9777     \bool_if:NF \g_@@_messages_for_Overleaf_bool
9778     { For~a~list~of~the~names~already~used,~type~H<return>. }
9779 }
9780 {
9781     The~names~already~defined~in~this~\@@_full_name_env:\ are:~
9782     \seq_use:Nnnn \g_@@_submatrix_names_seq { ~and~ } { ,~ } { ~and~ }.
9783 }

9784 \@@_msg_new:nn { r~or~l~with~preamble }
9785 {
9786     Erroneous~use.\\
9787     You~can't~use~the~key~'\l_keys_key_str'~in~your~\@@_full_name_env:~
9788     You~must~specify~the~alignment~of~your~columns~with~the~preamble~of~
9789     your~\@@_full_name_env:.\\
9790     This~key~will~be~ignored.
9791 }

9792 \@@_msg_new:nn { Hdotsfor~in~col~0 }
9793 {
9794     Erroneous~use.\\
9795     You~can't~use~\token_to_str:N \Hdotsfor\ in~an~exterior~column~of~
9796     the~array.~This~error~is~fatal.
9797 }

9798 \@@_msg_new:nn { bad~corner }

```

```

9799 {
9800   Bad-corner.\
9801   #1-is-an-incorrect-specification-for-a-corner~(in~the~key~
9802   'corners').~The~available~values~are:~NW,~SW,~NE~and~SE.\
9803   This~specification~of~corner~will~be~ignored.
9804 }
9805 \@@_msg_new:nn { bad-border }
9806 {
9807   Bad-border.\
9808   \l_keys_key_str\space-is-an-incorrect-specification-for-a-border~
9809   (in~the~key~'borders'~of~the~command~\token_to_str:N \Block).~
9810   The~available~values~are:~left,~right,~top~and~bottom~(and~you~can~
9811   also~use~the~key~'tikz'
9812   \IfPackageLoadedF { tikz }
9813   {~if~you~load~the~LaTeX~package~'tikz'}).\\
9814   This~specification~of~border~will~be~ignored.
9815 }
9816 \@@_msg_new:nn { TikzEveryCell~without~tikz }
9817 {
9818   TikZ~not~loaded.\
9819   You~can't~use~\token_to_str:N \TikzEveryCell\
9820   because~you~have~not~loaded~tikz.~
9821   This~command~will~be~ignored.
9822 }
9823 \@@_msg_new:nn { tikz-key~without~tikz }
9824 {
9825   TikZ~not~loaded.\
9826   You~can't~use~the~key~'tikz'~for~the~command~'\token_to_str:N
9827   \Block'~because~you~have~not~loaded~tikz.~
9828   This~key~will~be~ignored.
9829 }
9830 \@@_msg_new:nn { last-col~non~empty~for~NiceArray }
9831 {
9832   Erroneous~use.\
9833   In~the~\@@_full_name_env:,~you~must~use~the~key~
9834   'last-col'~without~value.\
9835   However,~you~can~go~on~for~this~time~
9836   (the~value~'\l_keys_value_tl'~will~be~ignored).
9837 }
9838 \@@_msg_new:nn { last-col~non~empty~for~NiceMatrixOptions }
9839 {
9840   Erroneous~use.\
9841   In~\token_to_str:N \NiceMatrixOptions,~you~must~use~the~key~
9842   'last-col'~without~value.\
9843   However,~you~can~go~on~for~this~time~
9844   (the~value~'\l_keys_value_tl'~will~be~ignored).
9845 }
9846 \@@_msg_new:nn { Block~too~large~1 }
9847 {
9848   Block~too~large.\
9849   You~try~to~draw~a~block~in~the~cell~#1-#2~of~your~matrix~but~the~matrix~is~
9850   too~small~for~that~block. \
9851   This~block~and~maybe~others~will~be~ignored.
9852 }
9853 \@@_msg_new:nn { Block~too~large~2 }
9854 {
9855   Block~too~large.\
9856   The~preamble~of~your~\@@_full_name_env:\ announces~\int_use:N
9857   \g_@@_static_num_of_col_int\
9858   columns~but~you~use~only~\int_use:N \c@jCol\ and~that's~why~a~block~
9859   specified~in~the~cell~#1-#2~can't~be~drawn.~You~should~add~some~ampersands~

```

```

9860      (&)~at~the~end~of~the~first~row~of~your~\@@_full_name_env:.\
9861      This~block~and~maybe~others~will~be~ignored.
9862  }

9863  \@@_msg_new:nn { unknown~column~type }
9864  {
9865      Bad~column~type.\
9866      The~column~type~'#1'~in~your~\@@_full_name_env:\
9867      is~unknown. \
9868      This~error~is~fatal.
9869  }

9870  \@@_msg_new:nn { unknown~column~type-S }
9871  {
9872      Bad~column~type.\
9873      The~column~type~'S'~in~your~\@@_full_name_env:\ is~unknown. \
9874      If~you~want~to~use~the~column~type~'S'~of~siunitx,~you~should~
9875      load~that~package. \
9876      This~error~is~fatal.
9877  }

9878  \@@_msg_new:nn { tabularnote~forbidden }
9879  {
9880      Forbidden~command.\
9881      You~can't~use~the~command~\token_to_str:N\tabularnote\
9882      ~here.~This~command~is~available~only~in~
9883      \{NiceTabular\},~\{NiceTabular*\}~and~\{NiceTabularX\}~or~in~
9884      the~argument~of~a~command~\token_to_str:N \caption\ included~
9885      in~an~environment~{table}. \
9886      This~command~will~be~ignored.
9887  }

9888  \@@_msg_new:nn { borders~forbidden }
9889  {
9890      Forbidden~key.\
9891      You~can't~use~the~key~'borders'~of~the~command~\token_to_str:N \Block\
9892      because~the~option~'rounded~corners'~
9893      is~in~force~with~a~non~zero~value.\
9894      This~key~will~be~ignored.
9895  }

9896  \@@_msg_new:nn { bottomrule~without~booktabs }
9897  {
9898      booktabs~not~loaded.\
9899      You~can't~use~the~key~'tabular/bottomrule'~because~you~haven't~
9900      loaded~'booktabs'.\
9901      This~key~will~be~ignored.
9902  }

9903  \@@_msg_new:nn { enumitem~not~loaded }
9904  {
9905      enumitem~not~loaded.\
9906      You~can't~use~the~command~\token_to_str:N\tabularnote\
9907      ~because~you~haven't~loaded~'enumitem'.\
9908      All~the~commands~\token_to_str:N\tabularnote\ will~be~
9909      ignored~in~the~document.
9910  }

9911  \@@_msg_new:nn { tikz~without~tikz }
9912  {
9913      Tikz~not~loaded.\
9914      You~can't~use~the~key~'tikz'~here~because~Tikz~is~not~
9915      loaded.~If~you~go~on,~that~key~will~be~ignored.
9916  }

9917  \@@_msg_new:nn { tikz~in~custom~line~without~tikz }
9918  {
9919      Tikz~not~loaded.\

```

```

9920     You-have-used-the-key~'tikz'~in-the-definition-of-a~
9921     customized~line~(with~'custom-line')~but~tikz-is-not-loaded.~
9922     You-can-go-on-but-you-will-have-another-error-if-you-actually~
9923     use-that~custom~line.
9924 }
9925 \@@_msg_new:nn { tikz-in-borders-without-tikz }
9926 {
9927     Tikz~not-loaded.\\
9928     You-have-used-the-key~'tikz'~in-a-key~'borders'~(of-a~
9929     command~'\token_to_str:N\Block')~but~tikz-is-not-loaded.~
9930     That~key~will-be-ignored.
9931 }
9932 \@@_msg_new:nn { without-color-inside }
9933 {
9934     If~order~to~use~\token_to_str:N \cellcolor,~\token_to_str:N \rowcolor,~
9935     \token_to_str:N \rowcolors\ or~\token_to_str:N \rowlistcolors\
9936     outside~\token_to_str:N \CodeBefore,~you~
9937     should~have~used~the~key~'color-inside'~in~your~\@@_full_name_env:~\\
9938     You-can-go-on-but-you-may-need-more~compilations.
9939 }
9940 \@@_msg_new:nn { color-in-custom-line-with-tikz }
9941 {
9942     Erroneous~use.\\
9943     In~a~'custom-line',~you~have~used~both~'tikz'~and~'color',~
9944     which~is~forbidden~(you~should~use~'color'~inside~the~key~'tikz').~
9945     The~key~'color'~will~be~discarded.
9946 }
9947 \@@_msg_new:nn { Wrong-last-row }
9948 {
9949     Wrong~number.\\
9950     You~have~used~'last-row=\int_use:N \l_@@_last_row_int'~but~your~
9951     \@@_full_name_env:\ seems~to~have~\int_use:N \c@iRow \ rows.~
9952     If~you~go~on,~the~value~of~\int_use:N \c@iRow \ will~be~used~for~
9953     last~row.~You~can~avoid~this~problem~by~using~'last-row'~
9954     without~value~(more~compilations~might~be~necessary).
9955 }
9956 \@@_msg_new:nn { Yet-in-env }
9957 {
9958     Nested~environments.\\
9959     Environments~of~nicematrix~can't~be~nested.\\
9960     This~error~is~fatal.
9961 }
9962 \@@_msg_new:nn { Outside-math-mode }
9963 {
9964     Outside~math~mode.\\
9965     The~\@@_full_name_env:\ can~be~used~only~in~math~mode~
9966     (and~not~in~\token_to_str:N \vcenter).\\
9967     This~error~is~fatal.
9968 }
9969 \@@_msg_new:nn { One-letter-allowed }
9970 {
9971     Bad~name.\\
9972     The~value~of~key~'\l_keys_key_str'~must~be~of~length~1.\\
9973     It~will~be~ignored.
9974 }
9975 \@@_msg_new:nn { TabularNote-in-CodeAfter }
9976 {
9977     Environment~{TabularNote}~forbidden.\\
9978     You~must~use~{TabularNote}~at~the~end~of~your~{NiceTabular}~
9979     but~*before*~the~\token_to_str:N \CodeAfter.\\

```

```

9980     This~environment~{TabularNote}~will~be~ignored.
9981 }
9982 \@@_msg_new:nn { varwidth~not~loaded }
9983 {
9984     varwidth~not~loaded.\\
9985     You~can't~use~the~column~type~'V'~because~'varwidth'~is~not~
9986     loaded.\\
9987     Your~column~will~behave~like~'p'.
9988 }
9989 \@@_msg_new:nnn { Unknow~key~for~RulesBis }
9990 {
9991     Unknow~key.\\
9992     Your~key~'\l_keys_key_str'~is~unknown~for~a~rule.\\
9993     \c_@@_available_keys_str
9994 }
9995 {
9996     The~available~keys~are~(in~alphabetic~order):~
9997     color,~
9998     dotted,~
9999     multiplicity,~
10000     sep-color,~
10001     tikz,~and~total~width.
10002 }
10003
10004 \@@_msg_new:nnn { Unknown~key~for~Block }
10005 {
10006     Unknown~key.\\
10007     The~key~'\l_keys_key_str'~is~unknown~for~the~command~\token_to_str:N
10008     \Block.\\ It~will~be~ignored. \\
10009     \c_@@_available_keys_str
10010 }
10011 {
10012     The~available~keys~are~(in~alphabetic~order):~&-in-blocks,~ampersand-in-blocks,~
10013     b,~B,~borders,~c,~draw,~fill,~hlines,~hvlines,~l,~line-width,~name,~
10014     opacity,~rounded-corners,~r,~respect-arraystretch,~t,~T,~tikz,~transparent~
10015     and~vlines.
10016 }
10017 \@@_msg_new:nnn { Unknown~key~for~Brace }
10018 {
10019     Unknown~key.\\
10020     The~key~'\l_keys_key_str'~is~unknown~for~the~commands~\token_to_str:N
10021     \UnderBrace\ and~\token_to_str:N \OverBrace.\\
10022     It~will~be~ignored. \\
10023     \c_@@_available_keys_str
10024 }
10025 {
10026     The~available~keys~are~(in~alphabetic~order):~color,~left-shorten,~
10027     right-shorten,~shorten~(which~fixes~both~left~shorten~and~
10028     right-shorten)~and~yshift.
10029 }
10030 \@@_msg_new:nnn { Unknown~key~for~CodeAfter }
10031 {
10032     Unknown~key.\\
10033     The~key~'\l_keys_key_str'~is~unknown.\\
10034     It~will~be~ignored. \\
10035     \c_@@_available_keys_str
10036 }
10037 {
10038     The~available~keys~are~(in~alphabetic~order):~
10039     delimiters/color,~
10040     rules~(with~the~subkeys~'color'~and~'width'),~
10041     sub-matrix~(several~subkeys)~

```

```

10042     and~xdots~(several~subkeys).~
10043     The~latter~is~for~the~command~\token_to_str:N \line.
10044 }

10045 \@@_msg_new:nnn { Unknown~key~for~CodeBefore }
10046 {
10047     Unknown~key.\\
10048     The~key~'\l_keys_key_str'~is~unknown.\\
10049     It~will~be~ignored. \\
10050     \c_@@_available_keys_str
10051 }
10052 {
10053     The~available~keys~are~(in~alphabetic~order):~
10054     create~cell~nodes,~
10055     delimiters/color~and~
10056     sub~matrix~(several~subkeys).
10057 }

10058 \@@_msg_new:nnn { Unknown~key~for~SubMatrix }
10059 {
10060     Unknown~key.\\
10061     The~key~'\l_keys_key_str'~is~unknown.\\
10062     That~key~will~be~ignored. \\
10063     \c_@@_available_keys_str
10064 }
10065 {
10066     The~available~keys~are~(in~alphabetic~order):~
10067     'delimiters/color',~
10068     'extra~height',~
10069     'hlines',~
10070     'hvlines',~
10071     'left~xshift',~
10072     'name',~
10073     'right~xshift',~
10074     'rules'~(with~the~subkeys~'color'~and~'width'),~
10075     'slim',~
10076     'vlines'~and~'xshift'~(which~sets~both~'left~xshift'~
10077     and~'right~xshift').\\
10078 }

10079 \@@_msg_new:nnn { Unknown~key~for~notes }
10080 {
10081     Unknown~key.\\
10082     The~key~'\l_keys_key_str'~is~unknown.\\
10083     That~key~will~be~ignored. \\
10084     \c_@@_available_keys_str
10085 }
10086 {
10087     The~available~keys~are~(in~alphabetic~order):~
10088     bottomrule,~
10089     code~after,~
10090     code~before,~
10091     detect~duplicates,~
10092     enumitem~keys,~
10093     enumitem~keys~para,~
10094     para,~
10095     label~in~list,~
10096     label~in~tabular~and~
10097     style.
10098 }

10099 \@@_msg_new:nnn { Unknown~key~for~RowStyle }
10100 {
10101     Unknown~key.\\
10102     The~key~'\l_keys_key_str'~is~unknown~for~the~command~
10103     \token_to_str:N \RowStyle. \\

```

```

10104     That~key~will~be~ignored. \\
10105     \c_@@_available_keys_str
10106 }
10107 {
10108     The~available~keys~are~(in~alphabetic~order):~
10109     'bold',~
10110     'cell-space-top-limit',~
10111     'cell-space-bottom-limit',~
10112     'cell-space-limits',~
10113     'color',~
10114     'nb-rows'~and~
10115     'rowcolor'.
10116 }
10117 \@@_msg_new:nnn { Unknown~key~for~NiceMatrixOptions }
10118 {
10119     Unknown~key.\\
10120     The~key~'\l_keys_key_str'~is~unknown~for~the~command~
10121     \token_to_str:N \NiceMatrixOptions. \\
10122     That~key~will~be~ignored. \\
10123     \c_@@_available_keys_str
10124 }
10125 {
10126     The~available~keys~are~(in~alphabetic~order):~
10127     &~in~blocks,~
10128     allow-duplicate-names,~
10129     ampersand-in-blocks,~
10130     caption-above,~
10131     cell-space-bottom-limit,~
10132     cell-space-limits,~
10133     cell-space-top-limit,~
10134     code-for-first-col,~
10135     code-for-first-row,~
10136     code-for-last-col,~
10137     code-for-last-row,~
10138     corners,~
10139     custom-key,~
10140     create-extra-nodes,~
10141     create-medium-nodes,~
10142     create-large-nodes,~
10143     custom-line,~
10144     delimiters~(several~subkeys),~
10145     end-of-row,~
10146     first-col,~
10147     first-row,~
10148     hlines,~
10149     hvlines,~
10150     hvlines-except-borders,~
10151     last-col,~
10152     last-row,~
10153     left-margin,~
10154     light-syntax,~
10155     light-syntax-expanded,~
10156     matrix/columns-type,~
10157     no-cell-nodes,~
10158     notes~(several~subkeys),~
10159     nullify-dots,~
10160     pgf-node-code,~
10161     renew-dots,~
10162     renew-matrix,~
10163     respect-arraystretch,~
10164     rounded-corners,~
10165     right-margin,~
10166     rules~(with~the~subkeys~'color'~and~'width'),~

```



```

10167     small,~
10168     sub-matrix~(several~subkeys),~
10169     vlines,~
10170     xdots~(several~subkeys).
10171 }

```

For '{NiceArray}', the set of keys is the same as for {NiceMatrix} excepted that there is no 1 and r.

```

10172 \@@_msg_new:nnn { Unknown~key~for~NiceArray }
10173 {
10174   Unknown~key.\\
10175   The~key~'\l_keys_key_str'~is~unknown~for~the~environment~
10176   \{NiceArray\}. \\
10177   That~key~will~be~ignored. \\
10178   \c_@@_available_keys_str
10179 }
10180 {
10181   The~available~keys~are~(in~alphabetic~order):~
10182   &-in-blocks,~
10183   ampersand-in-blocks,~
10184   b,~
10185   baseline,~
10186   c,~
10187   cell-space-bottom-limit,~
10188   cell-space-limits,~
10189   cell-space-top-limit,~
10190   code-after,~
10191   code-for-first-col,~
10192   code-for-first-row,~
10193   code-for-last-col,~
10194   code-for-last-row,~
10195   color-inside,~
10196   columns-width,~
10197   corners,~
10198   create-extra-nodes,~
10199   create-medium-nodes,~
10200   create-large-nodes,~
10201   extra-left-margin,~
10202   extra-right-margin,~
10203   first-col,~
10204   first-row,~
10205   hlines,~
10206   hvlines,~
10207   hvlines-except-borders,~
10208   last-col,~
10209   last-row,~
10210   left-margin,~
10211   light-syntax,~
10212   light-syntax-expanded,~
10213   name,~
10214   no-cell-nodes,~
10215   nullify-dots,~
10216   pgf-node-code,~
10217   renew-dots,~
10218   respect-arraystretch,~
10219   right-margin,~
10220   rounded-corners,~
10221   rules~(with~the~subkeys~'color'~and~'width'),~
10222   small,~
10223   t,~
10224   vlines,~
10225   xdots/color,~
10226   xdots/shorten-start,~
10227   xdots/shorten-end,~

```

```

10228     xdots/shorten-and~
10229     xdots/line-style.
10230 }

```

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`).

```

10231 \@@_msg_new:nnn { Unknown~key~for~NiceMatrix }
10232 {
10233   Unknown~key.\\
10234   The~key~'\l_keys_key_str'~is~unknown~for~the~
10235   \@@_full_name_env:. \\
10236   That~key~will~be~ignored. \\
10237   \c_@@_available_keys_str
10238 }
10239 {
10240   The~available~keys~are~(in~alphabetic~order):~
10241   &~in~blocks,~
10242   ampersand~in~blocks,~
10243   b,~
10244   baseline,~
10245   c,~
10246   cell-space-bottom-limit,~
10247   cell-space-limits,~
10248   cell-space-top-limit,~
10249   code~after,~
10250   code~for~first~col,~
10251   code~for~first~row,~
10252   code~for~last~col,~
10253   code~for~last~row,~
10254   color~inside,~
10255   columns~type,~
10256   columns~width,~
10257   corners,~
10258   create~extra~nodes,~
10259   create~medium~nodes,~
10260   create~large~nodes,~
10261   extra~left~margin,~
10262   extra~right~margin,~
10263   first~col,~
10264   first~row,~
10265   hlines,~
10266   hvlines,~
10267   hvlines~except~borders,~
10268   l,~
10269   last~col,~
10270   last~row,~
10271   left~margin,~
10272   light~syntax,~
10273   light~syntax~expanded,~
10274   name,~
10275   no~cell~nodes,~
10276   nullify~dots,~
10277   pgf~node~code,~
10278   r,~
10279   renew~dots,~
10280   respect~arraystretch,~
10281   right~margin,~
10282   rounded~corners,~
10283   rules~(with~the~subkeys~'color'~and~'width'),~
10284   small,~
10285   t,~
10286   vlimes,~
10287   xdots/color,~
10288   xdots/shorten~start,~

```

```

10289     xdots/shorten-end,~
10290     xdots/shorten-and~
10291     xdots/line-style.
10292 }
10293 \@@_msg_new:nnn { Unknown~key~for~NiceTabular }
10294 {
10295     Unknown~key.\\
10296     The~key~'\l_keys_key_str'~is~unknown~for~the~environment~
10297     \{NiceTabular\}. \\
10298     That~key~will~be~ignored. \\
10299     \c_@@_available_keys_str
10300 }
10301 {
10302     The~available~keys~are~(in~alphabetic~order):~
10303     &~in~blocks,~
10304     ampersand~in~blocks,~
10305     b,~
10306     baseline,~
10307     c,~
10308     caption,~
10309     cell-space-bottom-limit,~
10310     cell-space-limits,~
10311     cell-space-top-limit,~
10312     code-after,~
10313     code-for-first-col,~
10314     code-for-first-row,~
10315     code-for-last-col,~
10316     code-for-last-row,~
10317     color-inside,~
10318     columns-width,~
10319     corners,~
10320     custom-line,~
10321     create-extra-nodes,~
10322     create-medium-nodes,~
10323     create-large-nodes,~
10324     extra-left-margin,~
10325     extra-right-margin,~
10326     first-col,~
10327     first-row,~
10328     hlines,~
10329     hvlines,~
10330     hvlines-except-borders,~
10331     label,~
10332     last-col,~
10333     last-row,~
10334     left-margin,~
10335     light-syntax,~
10336     light-syntax-expanded,~
10337     name,~
10338     no-cell-nodes,~
10339     notes~(several~subkeys),~
10340     nullify-dots,~
10341     pgf-node-code,~
10342     renew-dots,~
10343     respect-arraystretch,~
10344     right-margin,~
10345     rounded-corners,~
10346     rules~(with~the~subkeys~'color'~and~'width'),~
10347     short-caption,~
10348     t,~
10349     tabularnote,~
10350     vlides,~
10351     xdots/color,~

```

```

10352     xdots/shorten-start,~
10353     xdots/shorten-end,~
10354     xdots/shorten-and~
10355     xdots/line-style.
10356 }
10357 \@@_msg_new:nnn { Duplicate-name }
10358 {
10359     Duplicate-name.\
10360     The-name~'\l_keys_value_tl'~is~already~used~and~you~shouldn't~use~
10361     the~same~environment~name~twice.~You~can~go~on,~but,~
10362     maybe,~you~will~have~incorrect~results~especially~
10363     if~you~use~'columns-width=auto'.~If~you~don't~want~to~see~this~
10364     message~again,~use~the~key~'allow-duplicate-names'~in~
10365     '\token_to_str:N \NiceMatrixOptions'.\
10366     \bool_if:NF \g_@@_messages_for_Overleaf_bool
10367     { For~a~list~of~the~names~already~used,~type~H~<return>. }
10368 }
10369 {
10370     The~names~already~defined~in~this~document~are:~
10371     \seq_use:Nnnn \g_@@_names_seq { ~and~ } { ,~ } { ~and~ }.
10372 }
10373 \@@_msg_new:nn { Option-auto-for-columns-width }
10374 {
10375     Erroneous~use.\
10376     You~can't~give~the~value~'auto'~to~the~key~'columns-width'~here.~
10377     That~key~will~be~ignored.
10378 }
10379 \@@_msg_new:nn { NiceTabularX~without~X }
10380 {
10381     NiceTabularX~without~X.\
10382     You~should~not~use~{NiceTabularX}~without~X~columns.\
10383     However,~you~can~go~on.
10384 }
10385 \@@_msg_new:nn { Preamble~forgotten }
10386 {
10387     Preamble~forgotten.\
10388     You~have~probably~forgotten~the~preamble~of~your~
10389     \@@_full_name_env:. \
10390     This~error~is~fatal.
10391 }

```

Contents

1	Declaration of the package and packages loaded	1
2	Security test	3
3	Collecting options	4
4	Technical definitions	5
5	Parameters	9
6	The command <code>\tabularnote</code>	20
7	Command for creation of rectangle nodes	24
8	The options	25
9	Important code used by <code>{NiceArrayWithDelims}</code>	36
10	The <code>\CodeBefore</code>	50
11	The environment <code>{NiceArrayWithDelims}</code>	53
12	We construct the preamble of the array	58
13	The redefinition of <code>\multicolumn</code>	74
14	The environment <code>{NiceMatrix}</code> and its variants	92
15	<code>{NiceTabular}</code> , <code>{NiceTabularX}</code> and <code>{NiceTabular*}</code>	93
16	After the construction of the array	94
17	We draw the dotted lines	101
18	The actual instructions for drawing the dotted lines with Tikz	114
19	User commands available in the new environments	120
20	The command <code>\line</code> accessible in code-after	126
21	The command <code>\RowStyle</code>	127
22	Colors of cells, rows and columns	130
23	The vertical and horizontal rules	142
24	The empty corners	157
25	The environment <code>{NiceMatrixBlock}</code>	160
26	The extra nodes	161
27	The blocks	165
28	How to draw the dotted lines transparently	189
29	Automatic arrays	189
30	The redefinition of the command <code>\dotfill</code>	190

31	The command <code>\diagbox</code>	191
32	The keyword <code>\CodeAfter</code>	192
33	The delimiters in the preamble	193
34	The command <code>\SubMatrix</code>	194
35	Les commandes <code>\UnderBrace</code> et <code>\OverBrace</code>	203
36	The command <code>TikzEveryCell</code>	206
37	The command <code>\ShowCellNames</code>	207
38	We process the options at package loading	210
39	About the package underscore	211
40	Error messages of the package	212