

The code of the package `nicematrix`*

F. Pantigny
fpantigny@wanadoo.fr

June 26, 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}
```

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

```
9 \RequirePackage { amsmath }
```

```
10 \RequirePackage { array }
```

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

```
11 \bool_const:Nn \c_@@_tagging_array_bool
12   { \IfPackageAtLeastTF { array } { 2024/05/01 } \c_true_bool \c_false_bool }
13 \bool_const:Nn \c_@@_testphase_table_bool
14   { \IfPackageLoadedTF { latex-lab-testphase-table } \c_true_bool \c_false_bool }
```

*This document corresponds to the version 6.28a of `nicematrix`, at the date of 2024/06/24.

```

15 \cs_new_protected:Npn \@@_error:n { \msg_error:nn { nicematrix } }
16 \cs_new_protected:Npn \@@_warning:n { \msg_warning:nn { nicematrix } }
17 \cs_new_protected:Npn \@@_error:nn { \msg_error:nnn { nicematrix } }
18 \cs_generate_variant:Nn \@@_error:nn { n e }
19 \cs_new_protected:Npn \@@_error:nnn { \msg_error:nnnn { nicematrix } }
20 \cs_new_protected:Npn \@@_fatal:n { \msg_fatal:nn { nicematrix } }
21 \cs_new_protected:Npn \@@_fatal:nn { \msg_fatal:nnn { nicematrix } }
22 \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).

```

23 \cs_new_protected:Npn \@@_msg_new:nnn #1 #2 #3
24 {
25   \bool_if:NTF \g_@@_messages_for_Overleaf_bool
26     { \msg_new:nnn { nicematrix } { #1 } { #2 \ \ #3 } }
27     { \msg_new:nnnn { nicematrix } { #1 } { #2 } { #3 } }
28 }

```

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

```

29 \cs_new_protected:Npn \@@_error_or_warning:n
30 { \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”.

```

31 \bool_new:N \g_@@_messages_for_Overleaf_bool
32 \bool_gset:Nn \g_@@_messages_for_Overleaf_bool
33 {
34   \str_if_eq_p:on \c_sys_jobname_str { _region_ } % for Emacs
35   || \str_if_eq_p:on \c_sys_jobname_str { output } % for Overleaf
36 }

```

```

37 \cs_new_protected:Npn \@@_msg_redirect_name:nn
38 { \msg_redirect_name:nnn { nicematrix } }
39 \cs_new_protected:Npn \@@_gredirect_none:n #1
40 {
41   \group_begin:
42   \globaldefs = 1
43   \@@_msg_redirect_name:nn { #1 } { none }
44   \group_end:
45 }
46 \cs_new_protected:Npn \@@_err_gredirect_none:n #1
47 {
48   \@@_error:n { #1 }
49   \@@_gredirect_none:n { #1 }
50 }
51 \cs_new_protected:Npn \@@_warning_gredirect_none:n #1
52 {
53   \@@_warning:n { #1 }
54   \@@_gredirect_none:n { #1 }
55 }

```

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

```

56 \cs_set:Npn \int_if_zero:NT #1 { \int_compare:nNnT #1 = \c_zero_int }
57 \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.

```
58 \@@_msg_new:nn { Internal~error }
59 {
60   Potential~problem~when~using~nicematrix.\\
61   The~package~nicematrix~have~detected~a~modification~of~the~
62   standard~environment~{array}~(of~the~package~array).~Maybe~you~will~encounter~
63   some~slight~problems~when~using~nicematrix.~If~you~don't~want~to~see~
64   this~message~again,~load~nicematrix~with:~\token_to_str:N
65   \usepackage[no-test-for-array]{nicematrix}.
66 }

67 \@@_msg_new:nn { mdwtab~loaded }
68 {
69   The~packages~'mdwtab'~and~'nicematrix'~are~incompatible.~
70   This~error~is~fatal.
71 }

72 \cs_new_protected:Npn \@@_security_test:n #1
73 {
74   \peek_meaning:NTF \ignorespaces
75   { \@@_security_test_i:w }
76   { \@@_error:n { Internal~error } }
77   #1
78 }

79 \bool_if:NTF \c_@@_tagging_array_bool
80 {
81   \cs_new_protected:Npn \@@_security_test_i:w \ignorespaces #1
82   {
83     \peek_meaning:NF \textonly@unskip { \@@_error:n { Internal~error } }
84     #1
85   }
86 }
87 {
88   \cs_new_protected:Npn \@@_security_test_i:w \ignorespaces #1
89   {
90     \peek_meaning:NF \unskip { \@@_error:n { Internal~error } }
91     #1
92   }
93 }
```

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

```
94 \hook_gput_code:nnn { begindocument / end } { . }
95 {
96   \IfPackageLoadedTF { mdwtab }
97   { \@@_fatal:n { mdwtab~loaded } }
98   {
99     \bool_if:NF \g_@@_no_test_for_array_bool
```

```

100      {
101      \group_begin:
102      \hbox_set:Nn \l_tmpa_box
103      {
104      \begin { tabular } { c > { \@@_security_test:n } c c }
105      text & & text
106      \end { tabular }
107      }
108      \group_end:
109      }
110  }
111  }

```

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`).

```

112 \cs_new_protected:Npn \@@_collect_options:n #1
113 {
114   \peek_meaning:NTF [
115     { \@@_collect_options:nw { #1 } }
116     { #1 { } }
117   }

```

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

```

118 \NewDocumentCommand \@@_collect_options:nw { m r[] }
119 { \@@_collect_options:nn { #1 } { #2 } }
120
121 \cs_new_protected:Npn \@@_collect_options:nn #1 #2
122 {
123   \peek_meaning:NTF [
124     { \@@_collect_options:nnw { #1 } { #2 } }
125     { #1 { #2 } }
126   }
127
128 \cs_new_protected:Npn \@@_collect_options:nnw #1#2[#3]
129 { \@@_collect_options:nn { #1 } { #2 , #3 } }

```

4 Technical definitions

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

```

130 \tl_const:Nn \c_@@_b_tl { b }
131 \tl_const:Nn \c_@@_c_tl { c }
132 \tl_const:Nn \c_@@_l_tl { l }

```

```

133 \tl_const:Nn \c_@@_r_tl { r }
134 \tl_const:Nn \c_@@_all_tl { all }
135 \tl_const:Nn \c_@@_dot_tl { . }
136 \tl_const:Nn \c_@@_default_tl { default }
137 \tl_const:Nn \c_@@_star_tl { * }
138 \str_const:Nn \c_@@_star_str { * }
139 \str_const:Nn \c_@@_r_str { r }
140 \str_const:Nn \c_@@_c_str { c }
141 \str_const:Nn \c_@@_l_str { l }
142 \str_const:Nn \c_@@_R_str { R }
143 \str_const:Nn \c_@@_C_str { C }
144 \str_const:Nn \c_@@_L_str { L }
145 \str_const:Nn \c_@@_j_str { j }
146 \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).

```

147 \tl_new:N \l_@@_argspec_tl
148 \cs_generate_variant:Nn \seq_set_split:Nnn { N o n }
149 \cs_generate_variant:Nn \str_lowercase:n { o }

150 \hook_gput_code:nnn { begindocument } { . }
151 {
152   \IfPackageLoadedTF { tikz }
153   {

```

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.

```

154   \tl_const:Nn \c_@@_pgfortikzpicture_tl { \exp_not:N \tikzpicture }
155   \tl_const:Nn \c_@@_endpgfortikzpicture_tl { \exp_not:N \endtikzpicture }
156 }
157 {
158   \tl_const:Nn \c_@@_pgfortikzpicture_tl { \exp_not:N \pgfpicture }
159   \tl_const:Nn \c_@@_endpgfortikzpicture_tl { \exp_not:N \endpgfpicture }
160 }
161 }

```

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`).

```

162 \IfClassLoadedTF { revtex4-1 }
163 { \bool_const:Nn \c_@@_revtex_bool \c_true_bool }
164 {
165   \IfClassLoadedTF { revtex4-2 }
166   { \bool_const:Nn \c_@@_revtex_bool \c_true_bool }
167   {

```

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

```

168   \cs_if_exist:NT \rvtx@ifformat@geq
169   { \bool_const:Nn \c_@@_revtex_bool \c_true_bool }
170   { \bool_const:Nn \c_@@_revtex_bool \c_false_bool }
171 }
172 }

```

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.

```

173 \cs_new_protected:Npn \@@_provide_pgfsyspdfmark:
174 {
175   \iow_now:Nn \@mainaux
176   {
177     \ExplSyntaxOn
178     \cs_if_free:NT \pgfsyspdfmark
179     { \cs_set_eq:NN \pgfsyspdfmark \gobblethree }
180     \ExplSyntaxOff
181   }
182   \cs_gset_eq:NN \@@_provide_pgfsyspdfmark: \prg_do_nothing:
183 }

```

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.

```

184 \ProvideDocumentCommand \iddots { }
185 {
186   \mathinner
187   {
188     \tex_mkern:D 1 mu
189     \box_move_up:nn { 1 pt } { \hbox { . } }
190     \tex_mkern:D 2 mu
191     \box_move_up:nn { 4 pt } { \hbox { . } }
192     \tex_mkern:D 2 mu
193     \box_move_up:nn { 7 pt }
194     { \vbox:n { \kern 7 pt \hbox { . } } }
195     \tex_mkern:D 1 mu
196   }
197 }

```

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.

```

198 \hook_gput_code:nnn { begindocument } { . }
199 {
200   \IfPackageLoadedTF { booktabs }
201   { \iow_now:Nn \@mainaux \nicematrix@redefine@check@rerun }
202   { }
203 }
204 \cs_set_protected:Npn \nicematrix@redefine@check@rerun
205 {
206   \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`).

```

207   \cs_set_protected:Npn \pgfutil@check@rerun ##1 ##2
208   {
209     \str_if_eq:eeF { nm- } { \tl_range:nnn { ##1 } 1 3 }
210     { \@@_old_pgfutil@check@rerun { ##1 } { ##2 } }
211   }
212 }

```

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

```

213 \hook_gput_code:nnn { begindocument } { . }

```

```

214 {
215   \IfPackageLoadedTF { colortbl }
216   { }
217 }

```

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.

```

218   \cs_set_protected:Npn \CT@arc@ { }
219   \cs_set_nopar:Npn \arrayrulecolor #1 # { \CT@arc@ { #1 } }
220   \cs_set_nopar:Npn \CT@arc@ #1 #2
221   {
222     \dim_compare:nNtT \baselineskip = \c_zero_dim \noalign
223     { \cs_gset_nopar:Npn \CT@arc@ { \color #1 { #2 } } }
224   }

```

Idem for `\CT@drs@`.

```

225   \cs_set_nopar:Npn \doublerulesepcolor #1 # { \CT@drs@ { #1 } }
226   \cs_set_nopar:Npn \CT@drs@ #1 #2
227   {
228     \dim_compare:nNtT \baselineskip = \c_zero_dim \noalign
229     { \cs_gset:Npn \CT@drsc@ { \color #1 { #2 } } }
230   }
231   \cs_set_nopar:Npn \hline
232   {
233     \noalign { \ifnum 0 = ` } \fi
234     \cs_set_eq:NN \hskip \vskip
235     \cs_set_eq:NN \vrule \hrule
236     \cs_set_eq:NN \@width \@height
237     { \CT@arc@ \vline }
238     \futurelet \reserved@a
239     \@xhline
240   }
241 }
242 }

```

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.

```

243 \cs_set_nopar:Npn \@@_standard_cline #1 { \@@_standard_cline:w #1 \q_stop }
244 \cs_set_nopar:Npn \@@_standard_cline:w #1-#2 \q_stop
245 {
246   \int_if_zero:nT \l_@@_first_col_int { \omit & }
247   \int_compare:nNtT { #1 } > \c_one_int
248   { \multispan { \int_eval:n { #1 - 1 } } & }
249   \multispan { \int_eval:n { #2 - #1 + 1 } }
250   {
251     \CT@arc@
252     \leaders \hrule \@height \arrayrulewidth \hfill

```

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

```

253   \skip_horizontal:N \c_zero_dim
254 }

```

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`.

```

255   \everycr { }
256   \cr
257   \noalign { \skip_vertical:N -\arrayrulewidth }
258 }

```

¹See question 99041 on TeX StackExchange.

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.

```
259 \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`.

```
260 { \@@_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*.

```
261 \cs_set:Npn \@@_cline_i:nn #1 #2 { \@@_cline_i:w #1|#2- \q_stop }
262 \cs_set:Npn \@@_cline_i:w #1|#2-#3 \q_stop
263 {
264   \tl_if_empty:nTF { #3 }
265     { \@@_cline_iii:w #1|#2-#2 \q_stop }
266     { \@@_cline_ii:w #1|#2-#3 \q_stop }
267 }
268 \cs_set:Npn \@@_cline_ii:w #1|#2-#3-\q_stop
269 { \@@_cline_iii:w #1|#2-#3 \q_stop }
270 \cs_set:Npn \@@_cline_iii:w #1|#2-#3 \q_stop
271 {
```

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).

```
272   \int_compare:nNnT { #1 } < { #2 }
273     { \multispan { \int_eval:n { #2 - #1 } } & }
274   \multispan { \int_eval:n { #3 - #2 + 1 } }
275   {
276     \CT@arc@
277     \leaders \hrule \@height \arrayrulewidth \hfill
278     \skip_horizontal:N \c_zero_dim
279   }
```

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

```
280   \peek_meaning_remove_ignore_spaces:NNTF \cline
281     { & \@@_cline_i:en { \int_eval:n { #3 + 1 } } }
282     { \everycr { } \cr }
283 }
284 \cs_generate_variant:Nn \@@_cline_i:nn { e n }
```

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

```
285 \cs_set_eq:NN \@@_math_toggle: \c_math_toggle_token
```

```
286 \cs_new_protected:Npn \@@_set_CT@arc@:n #1
287 {
288   \tl_if_blank:nF { #1 }
289   {
290     \tl_if_head_eq_meaning:nNTF { #1 } [
291       { \cs_set_nopar:Npn \CT@arc@ { \color #1 } }
292       { \cs_set_nopar:Npn \CT@arc@ { \color { #1 } } }
293     ]
294   }
295   \cs_generate_variant:Nn \@@_set_CT@arc@:n { o }
```

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


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

```

303 \cs_new:Npn \@@_exp_color_arg:Nn #1 #2
304 {
305   \tl_if_head_eq_meaning:nNTF { #2 } [
306     { #1 #2 }
307     { #1 { #2 } }
308   ]
309 \cs_generate_variant:Nn \@@_exp_color_arg:Nn { N o }

```

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

```

310 \cs_new_protected:Npn \@@_color:n #1
311 { \tl_if_blank:nF { #1 } { \@@_exp_color_arg:Nn \color { #1 } } }
312 \cs_generate_variant:Nn \@@_color:n { o }

```

```

313 \cs_new_protected:Npn \@@_rescan_for_spanish:N #1
314 {
315   \tl_set_rescan:Nno
316     #1
317     {
318       \char_set_catcode_other:N >
319       \char_set_catcode_other:N <
320     }
321   #1
322 }

```

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.

```

323 \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).

```

324 \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.

```

325 \NewExpandableDocumentCommand \NiceMatrixLastEnv { }
326 { \int_use:N \g_@@_env_int }

```

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

```

327 \cs_new_protected:Npn \@@_qpoint:n #1
328 { \pgfpointanchor { \@@_env: - #1 } { center } }

```

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

```

329 \bool_new:N \l_@@_tabular_bool

```

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

```
330 \bool_new:N \g_@@_delims_bool
331 \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.

```
332 \bool_new:N \l_@@_preamble_bool
333 \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.

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

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

```
335 \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.

```
336 \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).

```
337 \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.).

```
338 \dim_new:N \l_@@_col_width_dim
339 \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.

```
340 \int_new:N \g_@@_row_total_int
341 \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).

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

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

```
343 \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.

```
344 \tl_new:N \l_@@_hpos_cell_tl
345 \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`.

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

Idem for the mono-row blocks.

```
347 \dim_new:N \g_@@_blocks_ht_dim
348 \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}`).

```
349 \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`.

```
350 \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.

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

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

```
352 \bool_new:N \l_@@_notes_detect_duplicates_bool
353 \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.

```
354 \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).

```
355 \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).

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

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

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

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

```
358 \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.

```
359 \bool_new:N \l_@@_X_bool
360 \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 }`).

```
361 \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.

```
362 \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.

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

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

```
365 \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}`).

```
366 \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`).

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

For `\multicolumn`.

```
368 \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`.

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

```
370 \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 `:`.

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

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

```
373 \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`).

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

```
375 \cs_new_protected:Npn \@@_test_if_math_mode:
```

```
376 {
```

```
377   \if_mode_math: \else:
```

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

```
379   \fi:
```

```
380 }
```

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.

```
381 \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”.

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

```
383 \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`).

```
384 \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*.

```
385 \tl_new:N \g_@@_com_or_env_str
386 \tl_gset:Nn \g_@@_com_or_env_str { environment }

387 \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).

```
388 \cs_new:Npn \@@_full_name_env:
389 {
390   \str_if_eq:onTF \g_@@_com_or_env_str { command }
391   { command \space \c_backslash_str \g_@@_name_env_str }
392   { environment \space \{ \g_@@_name_env_str \} }
393 }
```

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

```
394 \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.

```
395 \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.

```
396 \tl_new:N \g_@@_pre_code_before_tl
397 \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.

```
398 \tl_new:N \g_@@_pre_code_after_tl
399 \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.

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

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

```
401 \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.

```
402 \int_new:N \l_@@_old_iRow_int
403 \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).

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

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

```
405 \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.

```
406 \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.

```
407 \bool_new:N \l_@@_X_columns_aux_bool
408 \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.

```
409 \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).

```
410 \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).

```
411 \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`).

```
412 \tl_new:N \l_@@_code_before_tl
413 \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).

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

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

```
415 \dim_new:N \l_@@_x_initial_dim
416 \dim_new:N \l_@@_y_initial_dim
417 \dim_new:N \l_@@_x_final_dim
418 \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.

```
419 \dim_new:N \l_@@_tmpc_dim
420 \dim_new:N \l_@@_tmpd_dim
```

```

421 \dim_new:N \g_@@_dp_row_zero_dim
422 \dim_new:N \g_@@_ht_row_zero_dim
423 \dim_new:N \g_@@_ht_row_one_dim
424 \dim_new:N \g_@@_dp_ante_last_row_dim
425 \dim_new:N \g_@@_ht_last_row_dim
426 \dim_new:N \g_@@_dp_last_row_dim

```

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

```

427 \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”.

```

428 \dim_new:N \g_@@_width_last_col_dim
429 \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.

```

430 \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!).

```

431 \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}`.

```

432 \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.

```

433 \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.

```

434 \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`).

```

435 \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.

```

436 \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).

```
437 \seq_new:N \g_@@_multicolumn_cells_seq
438 \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`).

```
439 \int_new:N \l_@@_row_min_int
440 \int_new:N \l_@@_row_max_int
441 \int_new:N \l_@@_col_min_int
442 \int_new:N \l_@@_col_max_int
```

The following counters will be used when drawing the rules.

```
443 \int_new:N \l_@@_start_int
444 \int_set_eq:NN \l_@@_start_int \c_one_int
445 \int_new:N \l_@@_end_int
446 \int_new:N \l_@@_local_start_int
447 \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.

```
448 \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).

```
449 \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`.

```
450 \tl_new:N \l_@@_fill_tl
451 \tl_new:N \l_@@_opacity_tl
452 \tl_new:N \l_@@_draw_tl
453 \seq_new:N \l_@@_tikz_seq
454 \clist_new:N \l_@@_borders_clist
455 \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.

```
456 \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`.

```
457 \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.

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

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

```
459 \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`).

```

460 \str_new:N \l_@@_hpos_block_str
461 \str_set:Nn \l_@@_hpos_block_str { c }
462 \bool_new:N \l_@@_hpos_of_block_cap_bool
463 \bool_new:N \l_@@_p_block_bool

```

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

```

464 \bool_new:N \l_@@_nocolor_used_bool

```

For the vertical position, the possible values are `c`, `t` and `b`.

```

465 \str_new:N \l_@@_vpos_block_str
466 \str_set:Nn \l_@@_vpos_block_str { c }

```

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

```

467 \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).

```

468 \bool_new:N \l_@@_vlines_block_bool
469 \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.

```

470 \int_new:N \g_@@_block_box_int

471 \dim_new:N \l_@@_submatrix_extra_height_dim
472 \dim_new:N \l_@@_submatrix_left_xshift_dim
473 \dim_new:N \l_@@_submatrix_right_xshift_dim
474 \clist_new:N \l_@@_hlines_clist
475 \clist_new:N \l_@@_vlines_clist
476 \clist_new:N \l_@@_submatrix_hlines_clist
477 \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}`).

```

478 \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.

```

479 \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`).

```

480 \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.

```

481 \int_new:N \l_@@_first_row_int
482 \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.

```
483 \int_new:N \l_@@_first_col_int
484 \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).

```
485 \int_new:N \l_@@_last_row_int
486 \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

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

Idem for `\l_@@_last_col_without_value_bool`

```
488 \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.

```
489 \int_new:N \l_@@_last_col_int
490 \int_set:Nn \l_@@_last_col_int { -2 }
```

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

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

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

```
491 \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`).

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

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

Some utilities

```

493 \cs_set_protected:Npn \@@_cut_on_hyphen:w #1-#2\q_stop
494 {
495   \cs_set_nopar:Npn \l_tmpa_tl { #1 }
496   \cs_set_nopar:Npn \l_tmpb_tl { #2 }
497 }

```

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.

```

498 \cs_new_protected:Npn \@@_expand_clist:N #1
499 {
500   \clist_if_in:NVF #1 \c_@@_all_tl
501   {
502     \clist_clear:N \l_tmpa_clist
503     \clist_map_inline:Nn #1
504     {
505       \tl_if_in:nnTF { ##1 } { - }
506       { \@@_cut_on_hyphen:w ##1 \q_stop }
507       {
508         \cs_set_nopar:Npn \l_tmpa_tl { ##1 }
509         \cs_set_nopar:Npn \l_tmpb_tl { ##1 }
510       }
511       \int_step_inline:nnn \l_tmpa_tl \l_tmpb_tl
512       { \clist_put_right:Nn \l_tmpa_clist { ####1 } }
513     }
514     \tl_set_eq:NN #1 \l_tmpa_clist
515   }
516 }

```

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.

```

517 \hook_gput_code:nnn { begindocument } { . }
518 {
519   \dim_const:Nn \c_@@_shift_Ldots_last_row_dim { 0.5 em }
520   \dim_const:Nn \c_@@_shift_exterior_Vdots_dim { 0.6 em }
521   \dim_const:Nn \c_@@_innersep_middle_dim { 0.17 em }
522 }

```

6 The command `\tabularnote`

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

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

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

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

```
523 \newcounter { tabularnote }
```

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

```
524 \int_new:N \g_@@_tabularnote_int
525 \cs_set:Npn \theHtabularnote { \int_use:N \g_@@_tabularnote_int }

526 \seq_new:N \g_@@_notes_seq
527 \seq_new:N \g_@@_notes_in_caption_seq
```

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

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

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

```
529 \seq_new:N \l_@@_notes_labels_seq
530 \newcounter{nicematrix_draft}
531 \cs_new_protected:Npn \@@_notes_format:n #1
532 {
533   \setcounter { nicematrix_draft } { #1 }
534   \@@_notes_style:n { nicematrix_draft }
535 }
```

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

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

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

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

```
537 \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`.

```
538 \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`.

```
539 \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`).

```
540 \hook_gput_code:nnn { begindocument } { . }
541 {
542   \IfPackageLoadedTF { enumitem }
543   {
```

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.

```
544     \newlist { tabularnotes } { enumerate } { 1 }
545     \setlist [ tabularnotes ]
546     {
547       topsep = 0pt ,
548       noitemsep ,
549       leftmargin = * ,
550       align = left ,
551       labelsep = 0pt ,
552       label =
553         \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotesi } } ,
554     }
555     \newlist { tabularnotes* } { enumerate* } { 1 }
556     \setlist [ tabularnotes* ]
557     {
558       afterlabel = \nobreak ,
559       itemjoin = \quad ,
560       label =
561         \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotes*i } }
562     }
```

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`.

```
563   \NewDocumentCommand \tabularnote { o m }
564   {
565     \bool_lazy_or:nnT { \cs_if_exist_p:N \@capttype } \l_@@_in_env_bool
566     {
567       \bool_lazy_and:nnTF { ! \l_@@_tabular_bool } \l_@@_in_env_bool
568       { \@@_error:n { tabularnote~forbidden } }
569       {
570         \bool_if:NTF \l_@@_in_caption_bool
571         \@@_tabularnote_caption:nn
572         \@@_tabularnote:nn
573         { #1 } { #2 }
574       }
575     }
```

```

576     }
577   }
578   {
579     \NewDocumentCommand \tabularnote { o m }
580     {
581       \@@_error_or_warning:n { enumitem~not~loaded }
582       \@@_gredirect_none:n { enumitem~not~loaded }
583     }
584   }
585 }

586 \cs_new_protected:Npn \@@_test_first_novalue:nnn #1 #2 #3
587 { \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`.

```

588 \cs_new_protected:Npn \@@_tabularnote:nn #1 #2
589 {

```

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).

```

590   \int_zero:N \l_tmpa_int
591   \bool_if:NT \l_@@_notes_detect_duplicates_bool
592   {

```

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`.

```

593   \int_zero:N \l_tmpb_int
594   \seq_map_indexed_inline:Nn \g_@@_notes_seq
595   {
596     \@@_test_first_novalue:nnn ##2 { \int_incr:N \l_tmpb_int }
597     \tl_if_eq:nnT { { #1 } { #2 } } { ##2 }
598     {
599       \tl_if_novalue:nTF { #1 }
600       { \int_set_eq:NN \l_tmpa_int \l_tmpb_int }
601       { \int_set:Nn \l_tmpa_int { ##1 } }
602       \seq_map_break:
603     }
604   }
605   \int_if_zero:nF \l_tmpa_int
606   { \int_add:Nn \l_tmpa_int \g_@@_notes_caption_int }
607 }
608 \int_if_zero:nT \l_tmpa_int
609 {
610   \seq_gput_right:Nn \g_@@_notes_seq { { #1 } { #2 } }
611   \tl_if_novalue:nT { #1 } { \int_gincr:N \c@tabularnote }
612 }
613 \seq_put_right:Nx \l_@@_notes_labels_seq
614 {
615   \tl_if_novalue:nTF { #1 }
616   {
617     \@@_notes_format:n
618     {
619       \int_eval:n

```

```

620      {
621      \int_if_zero:nTF \l_tmpa_int
622      \c@tabularnote
623      \l_tmpa_int
624      }
625    }
626  }
627  { #1 }
628 }
629 \peek_meaning:Nf \tabularnote
630 {

```

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`.

```

631   \hbox_set:Nn \l_tmpa_box
632   {

```

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

```

633   \@@_notes_label_in_tabular:n
634   {
635     \seq_use:Nnnn
636     \l_@@_notes_labels_seq { , } { , } { , }
637   }
638 }

```

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

```

639   \int_gdecr:N \c@tabularnote
640   \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.

```

641   \int_gincr:N \g_@@_tabularnote_int
642   \refstepcounter { tabularnote }
643   \int_compare:nNnT \l_tmpa_int = \c@tabularnote
644   { \int_gincr:N \c@tabularnote }
645   \seq_clear:N \l_@@_notes_labels_seq
646   \bool_lazy_or:nnTF
647   { \tl_if_eq_p:NN \l_@@_hpos_cell_tl \c_@@_c_tl }
648   { \tl_if_eq_p:NN \l_@@_hpos_cell_tl \c_@@_r_tl }
649   {
650     \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).

```

651   \skip_horizontal:n { \box_wd:N \l_tmpa_box }
652 }
653 { \box_use:N \l_tmpa_box }
654 }
655 }

```

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`.

```

656 \cs_new_protected:Npn \@@_tabularnote_caption:nn #1 #2
657 {
658   \bool_if:NTF \g_@@_caption_finished_bool
659   {

```

```

660      \int_compare:nNnT \c@tabulernote = \g_@@_notes_caption_int
661      { \int_gzero:N \c@tabulernote }

```

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

```

662      \seq_if_in:NnF \g_@@_notes_in_caption_seq { { #1 } { #2 } }
663      { \@@_error:n { Identical-notes-in-caption } }
664    }
665  {

```

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

```

666      \seq_if_in:NnTF \g_@@_notes_in_caption_seq { { #1 } { #2 } }
667      {

```

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

```

668      \bool_gset_true:N \g_@@_caption_finished_bool
669      \int_gset_eq:NN \g_@@_notes_caption_int \c@tabulernote
670      \int_gzero:N \c@tabulernote
671    }
672    { \seq_gput_right:Nn \g_@@_notes_in_caption_seq { { #1 } { #2 } } }
673  }

```

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!

```

674      \tl_if_novalue:nT { #1 } { \int_gincr:N \c@tabulernote }
675      \seq_put_right:Nx \l_@@_notes_labels_seq
676      {
677        \tl_if_novalue:nTF { #1 }
678        { \@@_notes_format:n { \int_use:N \c@tabulernote } }
679        { #1 }
680      }
681      \peek_meaning:NF \tabulernote
682      {
683        \@@_notes_label_in_tabular:n
684        { \seq_use:Nnnn \l_@@_notes_labels_seq { , } { , } { , } }
685        \seq_clear:N \l_@@_notes_labels_seq
686      }
687    }

688 \cs_new_protected:Npn \@@_count_novalue_first:nn #1 #2
689 { \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.

```

690 \cs_new_protected:Npn \@@_pgf_rect_node:nnnnn #1 #2 #3 #4 #5
691 {
692   \begin { pgfscope }
693   \pgfset
694   {
695     inner~sep = \c_zero_dim ,
696     minimum~size = \c_zero_dim
697   }
698   \pgftransformshift { \pgfpoint { 0.5 * ( #2 + #4 ) } { 0.5 * ( #3 + #5 ) } }
699   \pgfnode
700   { rectangle }

```



```

701     { center }
702     {
703         \vbox_to_ht:nn
704         { \dim_abs:n { #5 - #3 } }
705         {
706             \vfill
707             \hbox_to_wd:nn { \dim_abs:n { #4 - #2 } } { }
708         }
709     }
710     { #1 }
711     { }
712 \end { pgfscope }
713 }

```

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.

```

714 \cs_new_protected:Npn \@@_pgf_rect_node:nnn #1 #2 #3
715 {
716     \begin { pgfscope }
717     \pgfset
718     {
719         inner~sep = \c_zero_dim ,
720         minimum~size = \c_zero_dim
721     }
722     \pgftransformshift { \pgfpointscale { 0.5 } { \pgfpointadd { #2 } { #3 } } }
723     \pgfpointdiff { #3 } { #2 }
724     \pgfgetlastxy \l_tmpa_dim \l_tmpb_dim
725     \pgfnode
726     { rectangle }
727     { center }
728     {
729         \vbox_to_ht:nn
730         { \dim_abs:n \l_tmpb_dim }
731         { \vfill \hbox_to_wd:nn { \dim_abs:n \l_tmpa_dim } { } }
732     }
733     { #1 }
734     { }
735 \end { pgfscope }
736 }

```

8 The options

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

```

737 \tl_new:N \l_@@_caption_tl
738 \tl_new:N \l_@@_short_caption_tl
739 \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).

```

740 \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.

```

741 \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`.

```
742 \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`).

```
743 \dim_new:N \l_@@_cell_space_top_limit_dim
744 \dim_new:N \l_@@_cell_space_bottom_limit_dim
```

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

```
745 \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.

```
746 \dim_new:N \l_@@_xdots_inter_dim
747 \hook_gput_code:nnn { begindocument } { . }
748 { \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).

```
749 \dim_new:N \l_@@_xdots_shorten_start_dim
750 \dim_new:N \l_@@_xdots_shorten_end_dim
751 \hook_gput_code:nnn { begindocument } { . }
752 {
753   \dim_set:Nn \l_@@_xdots_shorten_start_dim { 0.3 em }
754   \dim_set:Nn \l_@@_xdots_shorten_end_dim { 0.3 em }
755 }
```

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.

```
756 \dim_new:N \l_@@_xdots_radius_dim
757 \hook_gput_code:nnn { begindocument } { . }
758 { \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.

```
759 \tl_new:N \l_@@_xdots_line_style_tl
760 \tl_const:Nn \c_@@_standard_tl { standard }
761 \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`.

```
762 \bool_new:N \l_@@_light_syntax_bool
763 \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).

```
764 \tl_new:N \l_@@_baseline_tl
765 \tl_set:Nn \l_@@_baseline_tl { c }
```

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

```
766 \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`).

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

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

```
768 \bool_new:N \l_@@_parallelize_diags_bool
769 \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.

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

```
771 \dim_new:N \l_@@_notes_above_space_dim
772 \hook_gput_code:nnn { begindocument } { . }
773 { \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.).

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

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

```
775 \cs_new_protected:Npn \@@_reset_arraystretch:
776 { \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).

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

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

```
778 \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.

```
779 \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”.

```
780 \bool_new:N \l_@@_medium_nodes_bool
781 \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).

```
782 \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`).

```
783 \dim_new:N \l_@@_left_margin_dim
784 \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`.

```
785 \dim_new:N \l_@@_extra_left_margin_dim
786 \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.

```
787 \tl_new:N \l_@@_end_of_row_tl
788 \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 “:”.

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

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

```
790 \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.

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

```
792 \keys_define:nn { NiceMatrix / xdots }
793 {
794   shorten-start .code:n =
795     \hook_gput_code:nnn { begindocument } { . }
796     { \dim_set:Nn \l_@@_xdots_shorten_start_dim { #1 } } ,
797   shorten-end .code:n =
798     \hook_gput_code:nnn { begindocument } { . }
799     { \dim_set:Nn \l_@@_xdots_shorten_end_dim { #1 } } ,
800   shorten-start .value_required:n = true ,
801   shorten-end .value_required:n = true ,
802   shorten .code:n =
803     \hook_gput_code:nnn { begindocument } { . }
804     {
805       \dim_set:Nn \l_@@_xdots_shorten_start_dim { #1 }
806       \dim_set:Nn \l_@@_xdots_shorten_end_dim { #1 }
807     } ,
808   shorten .value_required:n = true ,
809   horizontal-labels .bool_set:N = \l_@@_xdots_h_labels_bool ,
810   horizontal-labels .default:n = true ,
811   line-style .code:n =
812     {
813       \bool_lazy_or:nnTF
814         { \cs_if_exist_p:N \tikzpicture }
815         { \str_if_eq_p:nn { #1 } { standard } }
816         { \tl_set:Nn \l_@@_xdots_line_style_tl { #1 } }
817         { \@@_error:n { bad-option-for-line-style } }
818     } ,
```

```

819   line-style .value_required:n = true ,
820   color .tl_set:N = \l_@@_xdots_color_tl ,
821   color .value_required:n = true ,
822   radius .code:n =
823     \hook_gput_code:nnn { begindocument } { . }
824     { \dim_set:Nn \l_@@_xdots_radius_dim { #1 } } ,
825   radius .value_required:n = true ,
826   inter .code:n =
827     \hook_gput_code:nnn { begindocument } { . }
828     { \dim_set:Nn \l_@@_xdots_inter_dim { #1 } } ,
829   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 $\backslash\mathrm{tl_put_right:Nn}$ and not $\backslash\mathrm{tl_set:Nn}$ (or $\backslash\mathrm{tl_set:N}$) because we don't want a direct use of $\mathrm{up}=\dots$ erased by an absent $\wedge\{\dots\}$.

```

830   down .code:n = \tl_put_right:Nn \l_@@_xdots_down_tl { #1 } ,
831   up .code:n = \tl_put_right:Nn \l_@@_xdots_up_tl { #1 } ,
832   middle .code:n = \tl_put_right:Nn \l_@@_xdots_middle_tl { #1 } ,

```

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

```

833   draw-first .code:n = \prg_do_nothing: ,
834   unknown .code:n = \@@_error:n { Unknown~key~for~xdots }
835 }

```

```

836 \keys_define:nn { NiceMatrix / rules }
837 {
838   color .tl_set:N = \l_@@_rules_color_tl ,
839   color .value_required:n = true ,
840   width .dim_set:N = \arrayrulewidth ,
841   width .value_required:n = true ,
842   unknown .code:n = \@@_error:n { Unknown~key~for~rules }
843 }

```

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

```

844 \keys_define:nn { NiceMatrix / Global }
845 {
846   ampersand-in-blocks .bool_set:N = \l_@@_amp_in_blocks_bool ,
847   ampersand-in-blocks .default:n = true ,
848   &-in-blocks .meta:n = ampersand-in-blocks ,
849   no-cell-nodes .code:n =
850     \cs_set_protected:Npn \@@_node_for_cell:
851       { \box_use_drop:N \l_@@_cell_box } ,
852   no-cell-nodes .value_forbidden:n = true ,
853   rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
854   rounded-corners .default:n = 4 pt ,
855   custom-line .code:n = \@@_custom_line:n { #1 } ,
856   rules .code:n = \keys_set:nn { NiceMatrix / rules } { #1 } ,
857   rules .value_required:n = true ,
858   standard-cline .bool_set:N = \l_@@_standard_cline_bool ,
859   standard-cline .default:n = true ,
860   cell-space-top-limit .dim_set:N = \l_@@_cell_space_top_limit_dim ,
861   cell-space-top-limit .value_required:n = true ,
862   cell-space-bottom-limit .dim_set:N = \l_@@_cell_space_bottom_limit_dim ,
863   cell-space-bottom-limit .value_required:n = true ,
864   cell-space-limits .meta:n =
865     {
866       cell-space-top-limit = #1 ,
867       cell-space-bottom-limit = #1 ,
868     } ,

```

```

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

```

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

```

903     { \cs_set_eq:cN { @@ _ #1 } \@@_make_preamble_vlism:n }
904   }
905   { \@@_error:n { One~letter-allowed } }
906 } ,
907 vlines-in-sub-matrix .value_required:n = true ,
908 hvlines .code:n =
909   {
910     \bool_set_true:N \l_@@_hvlines_bool
911     \tl_set_eq:NN \l_@@_vlines_clist \c_@@_all_tl
912     \tl_set_eq:NN \l_@@_hlines_clist \c_@@_all_tl
913   } ,
914 hvlines-except-borders .code:n =
915   {
916     \tl_set_eq:NN \l_@@_vlines_clist \c_@@_all_tl
917     \tl_set_eq:NN \l_@@_hlines_clist \c_@@_all_tl
918     \bool_set_true:N \l_@@_hvlines_bool
919     \bool_set_true:N \l_@@_except_borders_bool
920   } ,
921 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.

```

922 renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
923 renew-dots .value_forbidden:n = true ,
924 nullify-dots .bool_set:N = \l_@@_nullify_dots_bool ,
925 create-medium-nodes .bool_set:N = \l_@@_medium_nodes_bool ,
926 create-large-nodes .bool_set:N = \l_@@_large_nodes_bool ,

```

```

927 create-extra-nodes .meta:n =
928   { create-medium-nodes , create-large-nodes } ,
929 left-margin .dim_set:N = \l_@@_left_margin_dim ,
930 left-margin .default:n = \arraycolsep ,
931 right-margin .dim_set:N = \l_@@_right_margin_dim ,
932 right-margin .default:n = \arraycolsep ,
933 margin .meta:n = { left-margin = #1 , right-margin = #1 } ,
934 margin .default:n = \arraycolsep ,
935 extra-left-margin .dim_set:N = \l_@@_extra_left_margin_dim ,
936 extra-right-margin .dim_set:N = \l_@@_extra_right_margin_dim ,
937 extra-margin .meta:n =
938   { extra-left-margin = #1 , extra-right-margin = #1 } ,
939 extra-margin .value_required:n = true ,
940 respect-arraystretch .code:n =
941   \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
942 respect-arraystretch .value_forbidden:n = true ,
943 pgf-node-code .tl_set:N = \l_@@_pgf_node_code_tl ,
944 pgf-node-code .value_required:n = true
945 }

```

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

```

946 \keys_define:nn { NiceMatrix / Env }
947 {
948   corners .clist_set:N = \l_@@_corners_clist ,
949   corners .default:n = { NW , SW , NE , SE } ,
950   code-before .code:n =
951     {
952       \tl_if_empty:nF { #1 }
953       {
954         \tl_gput_left:Nn \g_@@_pre_code_before_tl { #1 }
955         \bool_set_true:N \l_@@_code_before_bool
956       }
957     } ,
958   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}`.

```

959   c .code:n = \tl_set:Nn \l_@@_baseline_tl c ,
960   t .code:n = \tl_set:Nn \l_@@_baseline_tl t ,
961   b .code:n = \tl_set:Nn \l_@@_baseline_tl b ,
962   baseline .tl_set:N = \l_@@_baseline_tl ,
963   baseline .value_required:n = true ,
964   columns-width .code:n =
965     \tl_if_eq:nnTF { #1 } { auto }
966     { \bool_set_true:N \l_@@_auto_columns_width_bool }
967     { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,
968   columns-width .value_required:n = true ,
969   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.

```

970   \legacy_if:nF { measuring@ }
971   {
972     \str_set:Nx \l_tmpa_str { #1 }
973     \seq_if_in:NVTF \g_@@_names_seq \l_tmpa_str
974     { \@@_error:nn { Duplicate~name } { #1 } }
975     { \seq_gput_left:NV \g_@@_names_seq \l_tmpa_str }
976     \str_set_eq:NN \l_@@_name_str \l_tmpa_str
977   } ,
978   name .value_required:n = true ,
979   code-after .tl_gset:N = \g_nicematrix_code_after_tl ,

```

```

980     code-after .value_required:n = true ,
981     color-inside .code:n =
982       \bool_set_true:N \l_@@_color_inside_bool
983       \bool_set_true:N \l_@@_code_before_bool ,
984     color-inside .value_forbidden:n = true ,
985     colortbl-like .meta:n = color-inside
986   }
987 \keys_define:nn { NiceMatrix / notes }
988 {
989   para .bool_set:N = \l_@@_notes_para_bool ,
990   para .default:n = true ,
991   code-before .tl_set:N = \l_@@_notes_code_before_tl ,
992   code-before .value_required:n = true ,
993   code-after .tl_set:N = \l_@@_notes_code_after_tl ,
994   code-after .value_required:n = true ,
995   bottomrule .bool_set:N = \l_@@_notes_bottomrule_bool ,
996   bottomrule .default:n = true ,
997   style .cs_set:Np = \@@_notes_style:n #1 ,
998   style .value_required:n = true ,
999   label-in-tabular .cs_set:Np = \@@_notes_label_in_tabular:n #1 ,
1000   label-in-tabular .value_required:n = true ,
1001   label-in-list .cs_set:Np = \@@_notes_label_in_list:n #1 ,
1002   label-in-list .value_required:n = true ,
1003   enumitem-keys .code:n =
1004     {
1005       \hook_gput_code:nnn { begindocument } { . }
1006       {
1007         \IfPackageLoadedTF { enumitem }
1008           { \setlist* [ tabularnotes ] { #1 } }
1009           { }
1010       }
1011     } ,
1012   enumitem-keys .value_required:n = true ,
1013   enumitem-keys-para .code:n =
1014     {
1015       \hook_gput_code:nnn { begindocument } { . }
1016       {
1017         \IfPackageLoadedTF { enumitem }
1018           { \setlist* [ tabularnotes* ] { #1 } }
1019           { }
1020       }
1021     } ,
1022   enumitem-keys-para .value_required:n = true ,
1023   detect-duplicates .bool_set:N = \l_@@_notes_detect_duplicates_bool ,
1024   detect-duplicates .default:n = true ,
1025   unknown .code:n = \@@_error:n { Unknown~key~for~notes }
1026 }
1027 \keys_define:nn { NiceMatrix / delimiters }
1028 {
1029   max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1030   max-width .default:n = true ,
1031   color .tl_set:N = \l_@@_delimiters_color_tl ,
1032   color .value_required:n = true ,
1033 }

```

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

```

1034 \keys_define:nn { NiceMatrix }
1035 {
1036   NiceMatrixOptions .inherit:n =
1037     { NiceMatrix / Global } ,
1038   NiceMatrixOptions / xdots .inherit:n = NiceMatrix / xdots ,

```



```

1039 NiceMatrixOptions / rules .inherit:n = NiceMatrix / rules ,
1040 NiceMatrixOptions / notes .inherit:n = NiceMatrix / notes ,
1041 NiceMatrixOptions / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
1042 SubMatrix / rules .inherit:n = NiceMatrix / rules ,
1043 CodeAfter / xdots .inherit:n = NiceMatrix / xdots ,
1044 CodeBefore / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
1045 CodeAfter / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
1046 NiceMatrix .inherit:n =
1047 {
1048     NiceMatrix / Global ,
1049     NiceMatrix / Env ,
1050 } ,
1051 NiceMatrix / xdots .inherit:n = NiceMatrix / xdots ,
1052 NiceMatrix / rules .inherit:n = NiceMatrix / rules ,
1053 NiceTabular .inherit:n =
1054 {
1055     NiceMatrix / Global ,
1056     NiceMatrix / Env
1057 } ,
1058 NiceTabular / xdots .inherit:n = NiceMatrix / xdots ,
1059 NiceTabular / rules .inherit:n = NiceMatrix / rules ,
1060 NiceTabular / notes .inherit:n = NiceMatrix / notes ,
1061 NiceArray .inherit:n =
1062 {
1063     NiceMatrix / Global ,
1064     NiceMatrix / Env ,
1065 } ,
1066 NiceArray / xdots .inherit:n = NiceMatrix / xdots ,
1067 NiceArray / rules .inherit:n = NiceMatrix / rules ,
1068 pNiceArray .inherit:n =
1069 {
1070     NiceMatrix / Global ,
1071     NiceMatrix / Env ,
1072 } ,
1073 pNiceArray / xdots .inherit:n = NiceMatrix / xdots ,
1074 pNiceArray / rules .inherit:n = NiceMatrix / rules ,
1075 }

```

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

```

1076 \keys_define:nn { NiceMatrix / NiceMatrixOptions }
1077 {
1078     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1079     delimiters / color .value_required:n = true ,
1080     delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1081     delimiters / max-width .default:n = true ,
1082     delimiters .code:n = \keys_set:nn { NiceMatrix / delimiters } { #1 } ,
1083     delimiters .value_required:n = true ,
1084     width .dim_set:N = \l_@@_width_dim ,
1085     width .value_required:n = true ,
1086     last-col .code:n =
1087         \tl_if_empty:nF { #1 }
1088         { \@@_error:n { last-col~non-empty~for~NiceMatrixOptions } }
1089         \int_zero:N \l_@@_last_col_int ,
1090     small .bool_set:N = \l_@@_small_bool ,
1091     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.

```

1092     renew-matrix .code:n = \@@_renew_matrix: ,
1093     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}`.

```
1094     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.

```
1095     columns-width .code:n =
1096     \tl_if_eq:nnTF { #1 } { auto }
1097     { \@@_error:n { Option~auto~for~columns-width } }
1098     { \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.

```
1099     allow-duplicate-names .code:n =
1100     \@@_msg_redirect_name:nn { Duplicate~name } { none } ,
1101     allow-duplicate-names .value_forbidden:n = true ,
1102     notes .code:n = \keys_set:nn { NiceMatrix / notes } { #1 } ,
1103     notes .value_required:n = true ,
1104     sub-matrix .code:n = \keys_set:nn { NiceMatrix / sub-matrix } { #1 } ,
1105     sub-matrix .value_required:n = true ,
1106     matrix / columns-type .tl_set:N = \l_@@_columns_type_tl ,
1107     matrix / columns-type .value_required:n = true ,
1108     caption-above .bool_set:N = \l_@@_caption_above_bool ,
1109     caption-above .default:n = true ,
1110     unknown .code:n = \@@_error:n { Unknown~key~for~NiceMatrixOptions }
1111 } ,
```

`\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.

```
1112 \NewDocumentCommand \NiceMatrixOptions { m }
1113 { \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.

```
1114 \keys_define:nn { NiceMatrix / NiceMatrix }
1115 {
1116     last-col .code:n = \tl_if_empty:nTF { #1 }
1117     {
1118         \bool_set_true:N \l_@@_last_col_without_value_bool
1119         \int_set:Nn \l_@@_last_col_int { -1 }
1120     }
1121     { \int_set:Nn \l_@@_last_col_int { #1 } } ,
1122     columns-type .tl_set:N = \l_@@_columns_type_tl ,
1123     columns-type .value_required:n = true ,
1124     l .meta:n = { columns-type = l } ,
1125     r .meta:n = { columns-type = r } ,
1126     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1127     delimiters / color .value_required:n = true ,
1128     delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1129     delimiters / max-width .default:n = true ,
1130     delimiters .code:n = \keys_set:nn { NiceMatrix / delimiters } { #1 } ,
1131     delimiters .value_required:n = true ,
1132     small .bool_set:N = \l_@@_small_bool ,
1133     small .value_forbidden:n = true ,
1134     unknown .code:n = \@@_error:n { Unknown~key~for~NiceMatrix }
1135 }
```

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

```
1136 \keys_define:nn { NiceMatrix / NiceArray }
1137 {
```

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.

```
1138     small .bool_set:N = \l_@@_small_bool ,
1139     small .value_forbidden:n = true ,
1140     last-col .code:n = \tl_if_empty:nF { #1 }
1141         { \@@_error:n { last-col-non-empty-for-NiceArray } }
1142         \int_zero:N \l_@@_last_col_int ,
1143     r .code:n = \@@_error:n { r-or-l-with-preamble } ,
1144     l .code:n = \@@_error:n { r-or-l-with-preamble } ,
1145     unknown .code:n = \@@_error:n { Unknown-key-for-NiceArray }
1146 }

1147 \keys_define:nn { NiceMatrix / pNiceArray }
1148 {
1149     first-col .code:n = \int_zero:N \l_@@_first_col_int ,
1150     last-col .code:n = \tl_if_empty:nF { #1 }
1151         { \@@_error:n { last-col-non-empty-for-NiceArray } }
1152         \int_zero:N \l_@@_last_col_int ,
1153     first-row .code:n = \int_zero:N \l_@@_first_row_int ,
1154     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1155     delimiters / color .value_required:n = true ,
1156     delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1157     delimiters / max-width .default:n = true ,
1158     delimiters .code:n = \keys_set:nn { NiceMatrix / delimiters } { #1 } ,
1159     delimiters .value_required:n = true ,
1160     small .bool_set:N = \l_@@_small_bool ,
1161     small .value_forbidden:n = true ,
1162     r .code:n = \@@_error:n { r-or-l-with-preamble } ,
1163     l .code:n = \@@_error:n { r-or-l-with-preamble } ,
1164     unknown .code:n = \@@_error:n { Unknown-key-for-NiceMatrix }
1165 }
```

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

```
1166 \keys_define:nn { NiceMatrix / NiceTabular }
1167 {
```

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.

```
1168     width .code:n = \dim_set:Nn \l_@@_width_dim { #1 }
1169         \bool_set_true:N \l_@@_width_used_bool ,
1170     width .value_required:n = true ,
1171     notes .code:n = \keys_set:nn { NiceMatrix / notes } { #1 } ,
1172     tabularnote .tl_gset:N = \g_@@_tabularnote_tl ,
1173     tabularnote .value_required:n = true ,
1174     caption .tl_set:N = \l_@@_caption_tl ,
1175     caption .value_required:n = true ,
1176     short-caption .tl_set:N = \l_@@_short_caption_tl ,
1177     short-caption .value_required:n = true ,
1178     label .tl_set:N = \l_@@_label_tl ,
1179     label .value_required:n = true ,
1180     last-col .code:n = \tl_if_empty:nF { #1 }
1181         { \@@_error:n { last-col-non-empty-for-NiceArray } }
1182         \int_zero:N \l_@@_last_col_int ,
1183     r .code:n = \@@_error:n { r-or-l-with-preamble } ,
1184     l .code:n = \@@_error:n { r-or-l-with-preamble } ,
1185     unknown .code:n = \@@_error:n { Unknown-key-for-NiceTabular }
1186 }
```

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
```

```
1187 \keys_define:nn { NiceMatrix / CodeAfter }
1188 {
1189   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1190   delimiters / color .value_required:n = true ,
1191   rules .code:n = \keys_set:nn { NiceMatrix / rules } { #1 } ,
1192   rules .value_required:n = true ,
1193   xdots .code:n = \keys_set:nn { NiceMatrix / xdots } { #1 } ,
1194   sub-matrix .code:n = \keys_set:nn { NiceMatrix / sub-matrix } { #1 } ,
1195   sub-matrix .value_required:n = true ,
1196   unknown .code:n = \@@_error:n { Unknown~key~for~CodeAfter }
1197 }
```

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}`).

```
1198 \cs_new_protected:Npn \@@_cell_begin:w
1199 {
```

`\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.

```
1200   \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).

```
1201   \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
```

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

```
1202   \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.

```
1203   \int_compare:nNnT \c@jCol = \c_one_int
1204     { \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:`.

```
1205   \hbox_set:Nw \l_@@_cell_box
```

The following command is nullified in the tabulars.

```
1206   \@@_tuning_not_tabular_begin:
1207   \@@_tuning_first_row:
1208   \@@_tuning_last_row:
1209   \g_@@_row_style_tl
1210 }
```

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

```

1211 \cs_new_protected:Npn \@@_tuning_first_row:
1212 {
1213   \int_if_zero:nT \c@iRow
1214   {
1215     \int_compare:nNnT \c@jCol > \c_zero_int
1216     {
1217       \l_@@_code_for_first_row_tl
1218       \xglobal \colorlet { nicematrix-first-row } { . }
1219     }
1220   }
1221 }

```

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

```

1222 \cs_new_protected:Npn \@@_tuning_last_row:
1223 {
1224   \int_compare:nNnT \c@iRow = \l_@@_last_row_int
1225   {
1226     \l_@@_code_for_last_row_tl
1227     \xglobal \colorlet { nicematrix-last-row } { . }
1228   }
1229 }

```

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

```

1230 \cs_set_eq:NN \@@_tuning_key_small: \prg_do_nothing:

```

The following commands are nullified in the tabulars.

```

1231 \cs_set_nopar:Npn \@@_tuning_not_tabular_begin:
1232 {
1233   \c_math_toggle_token

```

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

```

1234   \@@_tuning_key_small:
1235 }
1236 \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.

```

1237 \cs_new_protected:Npn \@@_begin_of_row:
1238 {
1239   \int_gincr:N \c@iRow
1240   \dim_gset_eq:NN \g_@@_dp_ante_last_row_dim \g_@@_dp_last_row_dim
1241   \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
1242   \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1243   \pgfpicture
1244   \pgfrememberpicturerepositiononpagetrue
1245   \pgfcoordinate
1246   { \@@_env: - row - \int_use:N \c@iRow - base }
1247   { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
1248   \str_if_empty:NF \l_@@_name_str
1249   {
1250     \pgfnodealias
1251     { \l_@@_name_str - row - \int_use:N \c@iRow - base }
1252     { \@@_env: - row - \int_use:N \c@iRow - base }
1253   }
1254   \endpgfpicture
1255 }

```

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.

```

1256 \cs_new_protected:Npn \@@_update_for_first_and_last_row:
1257 {
1258   \int_if_zero:nTF \c@iRow
1259   {
1260     \dim_gset:Nn \g_@@_dp_row_zero_dim
1261     { \dim_max:nn \g_@@_dp_row_zero_dim { \box_dp:N \l_@@_cell_box } }
1262     \dim_gset:Nn \g_@@_ht_row_zero_dim
1263     { \dim_max:nn \g_@@_ht_row_zero_dim { \box_ht:N \l_@@_cell_box } }
1264   }
1265   {
1266     \int_compare:nNnT \c@iRow = \c_one_int
1267     {
1268       \dim_gset:Nn \g_@@_ht_row_one_dim
1269       { \dim_max:nn \g_@@_ht_row_one_dim { \box_ht:N \l_@@_cell_box } }
1270     }
1271   }
1272 }
1273 \cs_new_protected:Npn \@@_rotate_cell_box:
1274 {
1275   \box_rotate:Nn \l_@@_cell_box { 90 }
1276   \bool_if:NTF \g_@@_rotate_c_bool
1277   {
1278     \hbox_set:Nn \l_@@_cell_box
1279     {
1280       \c_math_toggle_token
1281       \vcenter { \box_use:N \l_@@_cell_box }
1282       \c_math_toggle_token
1283     }
1284   }
1285   {
1286     \int_compare:nNnT \c@iRow = \l_@@_last_row_int
1287     {
1288       \vbox_set_top:Nn \l_@@_cell_box
1289       {
1290         \vbox_to_zero:n { }
1291         \skip_vertical:n { - \box_ht:N \@arstrutbox + 0.8 ex }
1292         \box_use:N \l_@@_cell_box
1293       }
1294     }
1295   }
1296   \bool_gset_false:N \g_@@_rotate_bool
1297   \bool_gset_false:N \g_@@_rotate_c_bool
1298 }
1299 \cs_new_protected:Npn \@@_adjust_size_box:
1300 {
1301   \dim_compare:nNnT \g_@@_blocks_wd_dim > \c_zero_dim
1302   {
1303     \box_set_wd:Nn \l_@@_cell_box
1304     { \dim_max:nn { \box_wd:N \l_@@_cell_box } \g_@@_blocks_wd_dim }
1305     \dim_gzero:N \g_@@_blocks_wd_dim
1306   }
1307   \dim_compare:nNnT \g_@@_blocks_dp_dim > \c_zero_dim
1308   {
1309     \box_set_dp:Nn \l_@@_cell_box
1310     { \dim_max:nn { \box_dp:N \l_@@_cell_box } \g_@@_blocks_dp_dim }
1311     \dim_gzero:N \g_@@_blocks_dp_dim
1312   }
1313   \dim_compare:nNnT \g_@@_blocks_ht_dim > \c_zero_dim
1314   {

```

```

1315     \box_set_ht:Nn \l_@@_cell_box
1316     { \dim_max:nn { \box_ht:N \l_@@_cell_box } \g_@@_blocks_ht_dim }
1317     \dim_gzero:N \g_@@_blocks_ht_dim
1318   }
1319 }
1320 \cs_new_protected:Npn \@@_cell_end:
1321 {

```

The following command is nullified in the tabulars.

```

1322     \@@_tuning_not_tabular_end:
1323     \hbox_set_end:
1324     \@@_cell_end_i:
1325   }
1326 \cs_new_protected:Npn \@@_cell_end_i:
1327 {

```

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.

```

1328     \g_@@_cell_after_hook_tl
1329     \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
1330     \@@_adjust_size_box:
1331     \box_set_ht:Nn \l_@@_cell_box
1332     { \box_ht:N \l_@@_cell_box + \l_@@_cell_space_top_limit_dim }
1333     \box_set_dp:Nn \l_@@_cell_box
1334     { \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”).

```

1335     \@@_update_max_cell_width:

```

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

```

1336     \@@_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.

```

1337     \bool_if:NTF \g_@@_empty_cell_bool
1338     { \box_use_drop:N \l_@@_cell_box }
1339     {
1340       \bool_if:NTF \g_@@_not_empty_cell_bool
1341       \@@_node_for_cell:
1342       {
1343         \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > \c_zero_dim
1344         \@@_node_for_cell:

```

```

1345         { \box_use_drop:N \l_@@_cell_box }
1346     }
1347 }
1348 \int_gset:Nn \g_@@_col_total_int { \int_max:nn \g_@@_col_total_int \c@jCol }
1349 \bool_gset_false:N \g_@@_empty_cell_bool
1350 \bool_gset_false:N \g_@@_not_empty_cell_bool
1351 }

```

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

```

1352 \cs_new_protected:Npn \@@_update_max_cell_width:
1353 {
1354     \dim_gset:Nn \g_@@_max_cell_width_dim
1355     { \dim_max:nn \g_@@_max_cell_width_dim { \box_wd:N \l_@@_cell_box } }
1356 }

```

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`).

```

1357 \cs_new_protected:Npn \@@_cell_end_for_w_s:
1358 {
1359     \@@_math_toggle:
1360     \hbox_set_end:
1361     \bool_if:NF \g_@@_rotate_bool
1362     {
1363         \hbox_set:Nn \l_@@_cell_box
1364         {
1365             \makebox [ \l_@@_col_width_dim ] [ s ]
1366             { \hbox_unpack_drop:N \l_@@_cell_box }
1367         }
1368     }
1369     \@@_cell_end_i:
1370 }

```

```

1371 \pgfset
1372 {
1373     nicematrix / cell-node /.style =
1374     {
1375         inner-sep = \c_zero_dim ,
1376         minimum-width = \c_zero_dim
1377     }
1378 }

```

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

```

1379 \cs_new_protected:Npn \@@_node_for_cell:
1380 {
1381     \pgfpicture
1382     \pgfsetbaseline \c_zero_dim
1383     \pgfrememberpicturepositiononpagetrue
1384     \pgfset { nicematrix / cell-node }
1385     \pgfnode
1386     { rectangle }
1387     { base }
1388     {

```

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).

```

1389     \set@color
1390     \box_use_drop:N \l_@@_cell_box
1391 }
1392 { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1393 { \l_@@_pgf_node_code_tl }

```



```

1394 \str_if_empty:NF \l_@@_name_str
1395 {
1396   \pgfnodealias
1397   { \l_@@_name_str - \int_use:N \c@iRow - \int_use:N \c@jCol }
1398   { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1399 }
1400 \endpgfpicture
1401 }

```

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.

```

1402 \cs_new_protected:Npn \@@_patch_node_for_cell:n #1
1403 {
1404   \cs_new_protected:Npn \@@_patch_node_for_cell:
1405   {
1406     \hbox_set:Nn \l_@@_cell_box
1407     {
1408       \box_move_up:nn { \box_ht:N \l_@@_cell_box }
1409       \hbox_overlap_left:n
1410       {
1411         \pgfsys@markposition
1412         { \@@_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.

```

1413       #1
1414     }
1415     \box_use:N \l_@@_cell_box
1416     \box_move_down:nn { \box_dp:N \l_@@_cell_box }
1417     \hbox_overlap_left:n
1418     {
1419       \pgfsys@markposition
1420       { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - SE }
1421     }
1422   }
1423 }
1424 }
1425 }

```

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

```

1426 \bool_lazy_or:nnTF \sys_if_engine_xetex_p: \sys_if_output_dvi_p:
1427 {
1428   \@@_patch_node_for_cell:n
1429   { \skip_horizontal:n { 0.5 \box_wd:N \l_@@_cell_box } }
1430 }
1431 { \@@_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).

```
1432 \cs_new_protected:Npn \@@_instruction_of_type:nnn #1 #2 #3
1433 {
1434   \bool_if:nTF { #1 } \tl_gput_left:cx \tl_gput_right:cx
1435   { g_@@_ #2 _ lines _ tl }
1436   {
1437     \use:c { @@ _ draw _ #2 : nnn }
1438     { \int_use:N \c@iRow }
1439     { \int_use:N \c@jCol }
1440     { \exp_not:n { #3 } }
1441   }
1442 }
```

```
1443 \cs_new_protected:Npn \@@_array:
1444 {
1445   % \begin{macrocode}
1446   \dim_set:Nn \col@sep
1447   { \bool_if:nTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
1448   \dim_compare:nNnTF \l_@@_tabular_width_dim = \c_zero_dim
1449   { \cs_set_nopar:Npn \@halignto { } }
1450   { \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.

```
1451 \@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.

```
1452 [ \str_if_eq:onTF \l_@@_baseline_tl c c t ]
1453 }
```

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.

```
1454 \bool_if:nTF \c_@@_tagging_array_bool
1455 { \cs_set_eq:NN \@@_old_ar@ialign: \ar@ialign }
1456 { \cs_set_eq:NN \@@_old_ialign: \ialign }
```

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

```
1457 \cs_new_protected:Npn \@@_create_row_node:
1458 {
1459   \int_compare:nNnT \c@iRow > \g_@@_last_row_node_int
1460   {
1461     \int_gset_eq:NN \g_@@_last_row_node_int \c@iRow
1462     \@@_create_row_node_i:
1463   }
1464 }
1465 \cs_new_protected:Npn \@@_create_row_node_i:
1466 {
```

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

```
1467   \hbox
1468   {
1469     \bool_if:NT \l_@@_code_before_bool
1470     {
1471       \vtop
1472       {
```

```

1473         \skip_vertical:N 0.5\arrayrulewidth
1474         \pgfsys@markposition
1475         { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1476         \skip_vertical:N -0.5\arrayrulewidth
1477     }
1478 }
1479 \pgfpicture
1480 \pgfrememberpicturepositiononpagetrue
1481 \pgfcoordinate { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1482 { \pgfpoint \c_zero_dim { - 0.5 \arrayrulewidth } }
1483 \str_if_empty:NF \l_@@_name_str
1484 {
1485     \pgfnodealias
1486     { \l_@@_name_str - row - \int_eval:n { \c@iRow + 1 } }
1487     { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1488 }
1489 \endpgfpicture
1490 }
1491 }

```

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

```

1492 \cs_new:Npn \@@_everycr: { \noalign { \@@_everycr_i: } }

1493 \cs_new_protected:Npn \@@_everycr_i:
1494 {
1495     \bool_if:NT \c_@@_testphase_table_bool
1496     {
1497         \tbl_if_row_was_started:T { \UseTaggingSocket { tbl / row / end } }
1498         \tbl_update_cell_data_for_next_row:
1499     }
1500     \int_gzero:N \c@jCol
1501     \bool_gset_false:N \g_@@_after_col_zero_bool
1502     \bool_if:NF \g_@@_row_of_col_done_bool
1503     {
1504         \@@_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).

```

1505     \tl_if_empty:NF \l_@@_hlines_clist
1506     {
1507         \tl_if_eq:NnF \l_@@_hlines_clist \c_@@_all_tl
1508         {
1509             \exp_args:NNe
1510             \clist_if_in:NnT
1511             \l_@@_hlines_clist
1512             { \int_eval:n { \c@iRow + 1 } }
1513         }
1514     }

```

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.

```

1515     \int_compare:nNnT \c@iRow > { -1 }
1516     {
1517         \int_compare:nNnF \c@iRow = \l_@@_last_row_int
1518         { \hrule height \arrayrulewidth width \c_zero_dim }
1519     }
1520 }
1521 }
1522 }
1523 }

```

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

```

1524 \cs_set_protected:Npn \@@_renew_dots:
1525 {
1526   \cs_set_eq:NN \ldots \@@_Ldots
1527   \cs_set_eq:NN \cdots \@@_Cdots
1528   \cs_set_eq:NN \vdots \@@_Vdots
1529   \cs_set_eq:NN \ddots \@@_Ddots
1530   \cs_set_eq:NN \iddots \@@_Iddots
1531   \cs_set_eq:NN \dots \@@_Ldots
1532   \cs_set_eq:NN \hdotsfor \@@_Hdotsfor:
1533 }
1534 \cs_new_protected:Npn \@@_test_color_inside:
1535 {
1536   \bool_if:NF \l_@@_color_inside_bool
1537   {

```

We will issue an error only during the first run.

```

1538     \bool_if:NF \g_@@_aux_found_bool
1539     { \@@_error:n { without~color~inside } }
1540   }
1541 }

```

```

1542 \cs_new_protected:Npn \@@_redefine_everycr: { \everycr { \@@_everycr: } }
1543 \hook_gput_code:nnn { begindocument } { . }
1544 {
1545   \IfPackageLoadedTF { colortbl }
1546   {
1547     \cs_set_protected:Npn \@@_redefine_everycr:
1548     {
1549       \CT@everycr
1550       {
1551         \noalign { \cs_gset_eq:NN \CT@row@color \prg_do_nothing: }
1552         \@@_everycr:
1553       }
1554     }
1555   }
1556   { }
1557 }

```

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.

```

1558 \hook_gput_code:nnn { begindocument } { . }
1559 {
1560   \IfPackageLoadedTF { booktabs }
1561   {
1562     \cs_new_protected:Npn \@@_patch_booktabs:
1563     { \tl_put_left:Nn \@BTnormal \@@_create_row_node_i: }
1564   }
1565   { \cs_new_protected:Npn \@@_patch_booktabs: { } }
1566 }

```

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`.

```

1567 \cs_new_protected:Npn \@_some_initialization:
1568 {
1569   \dim_gset:Nn \g_@@_dp_row_zero_dim { \box_dp:N \@arstrutbox }
1570   \dim_gset:Nn \g_@@_ht_row_zero_dim { \box_ht:N \@arstrutbox }
1571   \dim_gset_eq:NN \g_@@_ht_row_one_dim \g_@@_ht_row_zero_dim
1572   \dim_gzero:N \g_@@_dp_ante_last_row_dim
1573   \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1574   \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
1575 }

```

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

```

1576 \cs_new_protected:Npn \@_pre_array_ii:
1577 {

```

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

```

1578   \int_gzero:N \g_@@_total_X_weight_int
1579   \@@_expand_clist:N \l_@@_hlines_clist
1580   \@@_expand_clist:N \l_@@_vlines_clist
1581   \@@_patch_booktabs:
1582   \box_clear_new:N \l_@@_cell_box
1583   \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}`).

```

1584   \bool_if:NT \l_@@_small_bool
1585   {
1586     \cs_set_nopar:Npn \arraystretch { 0.47 }
1587     \dim_set:Nn \arraycolsep { 1.45 pt }

```

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

```

1588     \cs_set_eq:NN \@_tuning_key_small: \scriptstyle
1589   }

```

```

1590   \bool_if:NT \g_@@_recreate_cell_nodes_bool
1591   {
1592     \tl_put_right:Nn \@_begin_of_row:
1593     {
1594       \pgfsys@markposition
1595       { \@_env: - row - \int_use:N \c@iRow - base }
1596     }
1597   }

```

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`.

```

1598   \bool_if:NTF \c_@@_tagging_array_bool
1599   {
1600     \cs_set_nopar:Npn \ar@ialign
1601     {
1602       \bool_if:NT \c_@@_testphase_table_bool \tbl_init_cell_data_for_table:
1603       \@@_redefine_everycr:
1604       \dim_zero:N \tabskip
1605       \@_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`.

```

1606         \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign:
1607         \halign
1608     }
1609 }

```

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).

```

1610 {
1611     \cs_set_nopar:Npn \ialign
1612     {
1613         \@@_redefine_everycr:
1614         \dim_zero:N \tabskip
1615         \@@_some_initialization:
1616         \cs_set_eq:NN \ialign \@@_old_ialign:
1617         \halign
1618     }
1619 }

```

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).

```

1620     \cs_set_eq:NN \@@_old_ldots \ldots
1621     \cs_set_eq:NN \@@_old_cdots \cdots
1622     \cs_set_eq:NN \@@_old_vdots \vdots
1623     \cs_set_eq:NN \@@_old_ddots \ddots
1624     \cs_set_eq:NN \@@_old_iddots \iddots
1625     \bool_if:NTF \l_@@_standard_cline_bool
1626     { \cs_set_eq:NN \cline \@@_standard_cline }
1627     { \cs_set_eq:NN \cline \@@_cline }
1628     \cs_set_eq:NN \Ldots \@@_Ldots
1629     \cs_set_eq:NN \Cdots \@@_Cdots
1630     \cs_set_eq:NN \Vdots \@@_Vdots
1631     \cs_set_eq:NN \Ddots \@@_Ddots
1632     \cs_set_eq:NN \Iddots \@@_Iddots
1633     \cs_set_eq:NN \Hline \@@_Hline:
1634     \cs_set_eq:NN \Hspace \@@_Hspace:
1635     \cs_set_eq:NN \Hdotsfor \@@_Hdotsfor:
1636     \cs_set_eq:NN \Vdotsfor \@@_Vdotsfor:
1637     \cs_set_eq:NN \Block \@@_Block:
1638     \cs_set_eq:NN \rotate \@@_rotate:
1639     \cs_set_eq:NN \OnlyMainNiceMatrix \@@_OnlyMainNiceMatrix:n
1640     \cs_set_eq:NN \dotfill \@@_dotfill:
1641     \cs_set_eq:NN \CodeAfter \@@_CodeAfter:
1642     \cs_set_eq:NN \diagbox \@@_diagbox:nn
1643     \cs_set_eq:NN \NotEmpty \@@_NotEmpty:
1644     \cs_set_eq:NN \RowStyle \@@_RowStyle:n
1645     \seq_map_inline:Nn \l_@@_custom_line_commands_seq
1646     { \cs_set_eq:cc { ##1 } { nicematrix - ##1 } }
1647     \cs_set_eq:NN \cellcolor \@@_cellcolor_tabular
1648     \cs_set_eq:NN \rowcolor \@@_rowcolor_tabular
1649     \cs_set_eq:NN \rowcolors \@@_rowcolors_tabular
1650     \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors_tabular
1651     \int_compare:nNnT \l_@@_first_row_int > \c_zero_int
1652     { \cs_set_eq:NN \@@_tuning_first_row: \prg_do_nothing: }
1653     \int_compare:nNnT \l_@@_last_row_int < \c_zero_int
1654     { \cs_set_eq:NN \@@_tuning_last_row: \prg_do_nothing: }
1655     \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:`.

```

1656     \cs_set_eq:NN \multicolumn \@@_multicolumn:nnn

```

```

1657 \hook_gput_code:nnn { env / tabular / begin } { nicematrix }
1658 { \cs_set_eq:NN \multicolumn \@_old_multicolumn }
1659 \@_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!).

```

1660 \tl_if_exist:NT \l_@@_note_in_caption_tl
1661 {
1662   \tl_if_empty:NF \l_@@_note_in_caption_tl
1663   {
1664     \int_gset_eq:NN \g_@@_notes_caption_int \l_@@_note_in_caption_tl
1665     \int_gset:Nn \c@tabularnote { \l_@@_note_in_caption_tl }
1666   }
1667 }

```

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).

```

1668 \seq_gclear:N \g_@@_multicolumn_cells_seq
1669 \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).

```

1670 \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`).

```

1671 \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.

```

1672 \int_gzero_new:N \g_@@_col_total_int
1673 \cs_set_eq:NN \@ifnextchar \new@ifnextchar
1674 \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.

```

1675 \tl_gclear_new:N \g_@@_Cdots_lines_tl
1676 \tl_gclear_new:N \g_@@_Ldots_lines_tl
1677 \tl_gclear_new:N \g_@@_Vdots_lines_tl
1678 \tl_gclear_new:N \g_@@_Ddots_lines_tl
1679 \tl_gclear_new:N \g_@@_Iddots_lines_tl
1680 \tl_gclear_new:N \g_@@_HVDotsfor_lines_tl

1681 \tl_gclear:N \g_nicematrix_code_before_tl
1682 \tl_gclear:N \g_@@_pre_code_before_tl
1683 }

```

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

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

```

1684 \cs_new_protected:Npn \@@_pre_array:
1685 {
1686   \cs_if_exist:NT \theiRow { \int_set_eq:NN \l_@@_old_iRow_int \c@iRow }
1687   \int_gzero_new:N \c@iRow
1688   \cs_if_exist:NT \thejCol { \int_set_eq:NN \l_@@_old_jCol_int \c@jCol }
1689   \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).

```

1690   \int_compare:nNnT \l_@@_last_row_int = { -1 }
1691   {
1692     \bool_set_true:N \l_@@_last_row_without_value_bool
1693     \bool_if:NT \g_@@_aux_found_bool
1694     { \int_set:Nn \l_@@_last_row_int { \seq_item:Nn \g_@@_size_seq 3 } }
1695   }
1696   \int_compare:nNnT \l_@@_last_col_int = { -1 }
1697   {
1698     \bool_if:NT \g_@@_aux_found_bool
1699     { \int_set:Nn \l_@@_last_col_int { \seq_item:Nn \g_@@_size_seq 6 } }
1700   }

```

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”.

```

1701   \int_compare:nNnT \l_@@_last_row_int > { -2 }
1702   {
1703     \tl_put_right:Nn \@@_update_for_first_and_last_row:
1704     {
1705       \dim_gset:Nn \g_@@_ht_last_row_dim
1706       { \dim_max:nn \g_@@_ht_last_row_dim { \box_ht:N \l_@@_cell_box } }
1707       \dim_gset:Nn \g_@@_dp_last_row_dim
1708       { \dim_max:nn \g_@@_dp_last_row_dim { \box_dp:N \l_@@_cell_box } }
1709     }
1710   }

1711   \seq_gclear:N \g_@@_cols_vlism_seq
1712   \seq_gclear:N \g_@@_submatrix_seq

```

Now the `\CodeBefore`.

```

1713   \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.

```

1714   \seq_gclear:N \g_@@_pos_of_blocks_seq

```

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

```

1715   \seq_gclear_new:N \g_@@_multicolumn_cells_seq
1716   \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.

```

1717   \int_gset:Nn \g_@@_last_row_node_int { -2 }

```

The value `-2` is important.

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

```

1718   \@@_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.

```
1719 \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]`).

```
1720 \dim_zero_new:N \l_@@_left_delim_dim
1721 \dim_zero_new:N \l_@@_right_delim_dim
1722 \bool_if:NTF \g_@@_delims_bool
1723 {
```

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

```
1724 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_left_delim_tl $ }
1725 \dim_set:Nn \l_@@_left_delim_dim { \box_wd:N \l_tmpa_box }
1726 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_right_delim_tl $ }
1727 \dim_set:Nn \l_@@_right_delim_dim { \box_wd:N \l_tmpa_box }
1728 }
1729 {
1730 \dim_gset:Nn \l_@@_left_delim_dim
1731 { 2 \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
1732 \dim_gset_eq:NN \l_@@_right_delim_dim \l_@@_left_delim_dim
1733 }
```

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).

```
1734 \hbox_set:Nw \l_@@_the_array_box
1735 \bool_if:NT \c_@@_testphase_table_bool
1736 { \UseTaggingSocket { tbl / hmode / begin } }
1737 \skip_horizontal:N \l_@@_left_margin_dim
1738 \skip_horizontal:N \l_@@_extra_left_margin_dim
1739 \c_math_toggle_token
1740 \bool_if:NTF \l_@@_light_syntax_bool
1741 { \use:c { @@-light-syntax } }
1742 { \use:c { @@-normal-syntax } }
1743 }
```

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

```
1744 \cs_new_protected_nopar:Npn \@@_CodeBefore_Body:w #1 \Body
1745 {
1746 \tl_set:Nn \l_tmpa_tl { #1 }
1747 \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
1748 { \@@_rescan_for_spanish:N \l_tmpa_tl }
1749 \tl_gput_left:Nv \g_@@_pre_code_before_tl \l_tmpa_tl
1750 \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...

```
1751 \@@_pre_array:
1752 }
```

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).

```
1753 \cs_new_protected:Npn \@@_pre_code_before:
1754 {
```

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).

```
1755 \int_set:Nn \c@iRow { \seq_item:Nn \g_@@_size_seq 2 }
1756 \int_set:Nn \c@jCol { \seq_item:Nn \g_@@_size_seq 5 }
1757 \int_set_eq:NN \g_@@_row_total_int { \seq_item:Nn \g_@@_size_seq 3 }
1758 \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.

```
1759 \pgfsys@markposition { \@@_env: - position }
1760 \pgfsys@getposition { \@@_env: - position } \@@_picture_position:
1761 \pgfpicture
1762 \pgf@relevantforpicturesizefalse
```

First, the recreation of the `row` nodes.

```
1763 \int_step_inline:nnn \l_@@_first_row_int { \g_@@_row_total_int + 1 }
1764 {
1765   \pgfsys@getposition { \@@_env: - row - ##1 } \@@_node_position:
1766   \pgfcoordinate { \@@_env: - row - ##1 }
1767   { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1768 }
```

Now, the recreation of the `col` nodes.

```
1769 \int_step_inline:nnn \l_@@_first_col_int { \g_@@_col_total_int + 1 }
1770 {
1771   \pgfsys@getposition { \@@_env: - col - ##1 } \@@_node_position:
1772   \pgfcoordinate { \@@_env: - col - ##1 }
1773   { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1774 }
```

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

```
1775 \@@_create_diag_nodes:
```

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

```
1776 \bool_if:NT \g_@@_recreate_cell_nodes_bool \@@_recreate_cell_nodes:
1777 \endpgfpicture
```

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

```
1778 \@@_create_blocks_nodes:
1779 \IfPackageLoadedTF { tikz }
1780 {
1781   \tikzset
1782   {
1783     every-picture / .style =
1784     { overlay , name-prefix = \@@_env: - }
1785   }
1786 }
1787 { }
1788 \cs_set_eq:NN \cellcolor \@@_cellcolor
1789 \cs_set_eq:NN \rectanglecolor \@@_rectanglecolor
1790 \cs_set_eq:NN \roundedrectanglecolor \@@_roundedrectanglecolor
1791 \cs_set_eq:NN \rowcolor \@@_rowcolor
```

```

1792 \cs_set_eq:NN \rowcolors \@@_rowcolors
1793 \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors
1794 \cs_set_eq:NN \arraycolor \@@_arraycolor
1795 \cs_set_eq:NN \columncolor \@@_columncolor
1796 \cs_set_eq:NN \chessboardcolors \@@_chessboardcolors
1797 \cs_set_eq:NN \SubMatrix \@@_SubMatrix_in_code_before
1798 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
1799 \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
1800 }

```

```

1801 \cs_new_protected:Npn \@@_exec_code_before:
1802 {
1803   \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`.

```

1804 \@@_add_to_colors_seq:nn { { nocolor } } { }
1805 \bool_gset_false:N \g_@@_recreate_cell_nodes_bool
1806 \group_begin:

```

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

```

1807 \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`).

```

1808 \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
1809 { \@@_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`.

```

1810 \exp_last_unbraced:NV \@@_CodeBefore_keys:
1811 \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.

```

1812 \@@_actually_color:
1813 \l_@@_code_before_tl
1814 \q_stop
1815 \bool_if:NT \l_@@_tabular_bool \c_math_toggle_token
1816 \group_end:
1817 \bool_if:NT \g_@@_recreate_cell_nodes_bool
1818 { \tl_put_left:Nn \@@_node_for_cell: \@@_patch_node_for_cell: }
1819 }

```

```

1820 \keys_define:nn { NiceMatrix / CodeBefore }
1821 {
1822   create-cell-nodes .bool_gset:N = \g_@@_recreate_cell_nodes_bool ,
1823   create-cell-nodes .default:n = true ,
1824   sub-matrix .code:n = \keys_set:nn { NiceMatrix / sub-matrix } { #1 } ,
1825   sub-matrix .value_required:n = true ,
1826   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1827   delimiters / color .value_required:n = true ,
1828   unknown .code:n = \@@_error:n { Unknown~key~for~CodeBefore }
1829 }

```

```

1830 \NewDocumentCommand \@@_CodeBefore_keys: { 0 { } }
1831 {
1832   \keys_set:nn { NiceMatrix / CodeBefore } { #1 }
1833   \@@_CodeBefore:w
1834 }

```

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.

```

1835 \cs_new_protected:Npn \@@_CodeBefore:w #1 \q_stop
1836 {
1837   \bool_if:NT \g_@@_aux_found_bool
1838   {
1839     \@@_pre_code_before:
1840     #1
1841   }
1842 }

```

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.

```

1843 \cs_new_protected:Npn \@@_recreate_cell_nodes:
1844 {
1845   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
1846   {
1847     \pgfsys@getposition { \@@_env: - ##1 - base } \@@_node_position:
1848     \pgfcoordinate { \@@_env: - row - ##1 - base }
1849     { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1850     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
1851     {
1852       \cs_if_exist:cT
1853       { pgf @ sys @ pdf @ mark @ pos @ \@@_env: - ##1 - #####1 - NW }
1854       {
1855         \pgfsys@getposition
1856         { \@@_env: - ##1 - #####1 - NW }
1857         \@@_node_position:
1858         \pgfsys@getposition
1859         { \@@_env: - ##1 - #####1 - SE }
1860         \@@_node_position_i:
1861         \@@_pgf_rect_node:nnn
1862         { \@@_env: - ##1 - #####1 }
1863         { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1864         { \pgfpointdiff \@@_picture_position: \@@_node_position_i: }
1865       }
1866     }
1867   }
1868   \int_step_inline:nn \c@iRow
1869   {
1870     \pgfnodealias
1871     { \@@_env: - ##1 - last }
1872     { \@@_env: - ##1 - \int_use:N \c@jCol }
1873   }
1874   \int_step_inline:nn \c@jCol
1875   {
1876     \pgfnodealias
1877     { \@@_env: - last - ##1 }
1878     { \@@_env: - \int_use:N \c@iRow - ##1 }
1879   }
1880   \@@_create_extra_nodes:
1881 }

```

```

1882 \cs_new_protected:Npn \@@_create_blocks_nodes:
1883 {
1884   \pgfpicture
1885   \pgf@relevantforpicturesizefalse
1886   \pgfrememberpicturepositiononpagetrue
1887   \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
1888   { \@@_create_one_block_node:nnnnn ##1 }
1889   \endpgfpicture
1890 }

```

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

```

1891 \cs_new_protected:Npn \@@_create_one_block_node:nnnnn #1 #2 #3 #4 #5
1892 {
1893   \tl_if_empty:nF { #5 }
1894   {
1895     \@@_qpoint:n { col - #2 }
1896     \dim_set_eq:NN \l_tmpa_dim \pgf@x
1897     \@@_qpoint:n { #1 }
1898     \dim_set_eq:NN \l_tmpb_dim \pgf@y
1899     \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
1900     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
1901     \@@_qpoint:n { \int_eval:n { #3 + 1 } }
1902     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
1903     \@@_pgf_rect_node:nnnnn
1904     { \@@_env: - #5 }
1905     { \dim_use:N \l_tmpa_dim }
1906     { \dim_use:N \l_tmpb_dim }
1907     { \dim_use:N \l_@@_tmpc_dim }
1908     { \dim_use:N \l_@@_tmpd_dim }
1909   }
1910 }

1911 \cs_new_protected:Npn \@@_patch_for_revtext:
1912 {
1913   \cs_set_eq:NN \@addamp \@addamp@LaTeX
1914   \cs_set_eq:NN \insert@column \insert@column@array
1915   \cs_set_eq:NN \@classx \@classx@array
1916   \cs_set_eq:NN \@xarraycr \@xarraycr@array
1917   \cs_set_eq:NN \@arraycr \@arraycr@array
1918   \cs_set_eq:NN \@xargarraycr \@xargarraycr@array
1919   \cs_set_eq:NN \array \array@array
1920   \cs_set_eq:NN \@array \@array@array
1921   \cs_set_eq:NN \@tabular \@tabular@array
1922   \cs_set_eq:NN \@mkpream \@mkpream@array
1923   \cs_set_eq:NN \endarray \endarray@array
1924   \cs_set:Npn \@tabarray { \ifnextchar [ { \@array } { \@array [ c ] } }
1925   \cs_set:Npn \endtabular { \endarray $\egroup} % $
1926 }

```

11 The environment `{NiceArrayWithDelims}`

```

1927 \NewDocumentEnvironment { NiceArrayWithDelims }
1928 { m m 0 { } m ! 0 { } t \CodeBefore }
1929 {

```

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).

```

1930 \bool_if:NT \c_@@_revtex_bool \@@_patch_for_revtex:
1931 \@@_provide_pgfsyspdfmark:
1932 \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.

```

1933 \bgroup

1934 \tl_gset:Nn \g_@@_left_delim_tl { #1 }
1935 \tl_gset:Nn \g_@@_right_delim_tl { #2 }
1936 \tl_gset:Nn \g_@@_user_preamble_tl { #4 }
1937 \tl_if_empty:NT \g_@@_user_preamble_tl { \@@_fatal:n { empty-preamble } }

1938 \int_gzero:N \g_@@_block_box_int
1939 \dim_zero:N \g_@@_width_last_col_dim
1940 \dim_zero:N \g_@@_width_first_col_dim
1941 \bool_gset_false:N \g_@@_row_of_col_done_bool
1942 \str_if_empty:NT \g_@@_name_env_str
1943 { \str_gset:Nn \g_@@_name_env_str { NiceArrayWithDelims } }
1944 \bool_if:NTF \l_@@_tabular_bool
1945 \mode_leave_vertical:
1946 \@@_test_if_math_mode:
1947 \bool_if:NT \l_@@_in_env_bool { \@@_fatal:n { Yet~in~env } }
1948 \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.

```

1949 \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.

```

1950 \cs_if_exist:NT \tikz@library@external@loaded
1951 {
1952 \tikzexternaldisable
1953 \cs_if_exist:NT \ifstandalone
1954 { \tikzset { external / optimize = false } }
1955 }

```

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

```

1956 \int_gincr:N \g_@@_env_int
1957 \bool_if:NF \l_@@_block_auto_columns_width_bool
1958 { \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`).

```

1959 \seq_gclear:N \g_@@_blocks_seq
1960 \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`.

```

1961 \seq_gclear:N \g_@@_pos_of_stroken_blocks_seq
1962 \seq_gclear:N \g_@@_pos_of_xdots_seq
1963 \tl_gclear_new:N \g_@@_code_before_tl
1964 \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}`

```

1965 \tl_if_exist:cTF { c_@@ _ \int_use:N \g_@@_env_int _ tl }
1966 {
1967     \bool_gset_true:N \g_@@_aux_found_bool
1968     \use:c { c_@@ _ \int_use:N \g_@@_env_int _ tl }
1969 }
1970 { \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.

```

1971 \tl_gclear:N \g_@@_aux_tl
1972 \tl_if_empty:NF \g_@@_code_before_tl
1973 {
1974     \bool_set_true:N \l_@@_code_before_bool
1975     \tl_put_right:NV \l_@@_code_before_tl \g_@@_code_before_tl
1976 }
1977 \tl_if_empty:NF \g_@@_pre_code_before_tl
1978 { \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.

```

1979 \bool_if:NTF \g_@@_delims_bool
1980 { \keys_set:nn { NiceMatrix / pNiceArray } }
1981 { \keys_set:nn { NiceMatrix / NiceArray } }
1982 { #3 , #5 }

```

```

1983 \@@_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:.

```

1984 \bool_if:nTF { #6 } \@@_CodeBefore_Body:w \@@_pre_array:
1985 }

```

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

```

1986 {
1987     \bool_if:NTF \l_@@_light_syntax_bool
1988     { \use:c { end @@-light-syntax } }
1989     { \use:c { end @@-normal-syntax } }
1990     \c_math_toggle_token
1991     \skip_horizontal:N \l_@@_right_margin_dim
1992     \skip_horizontal:N \l_@@_extra_right_margin_dim
1993
1994     %% awful workaround
1995     \int_compare:nNnT \g_@@_col_total_int = \c_one_int
1996     {
1997         \dim_compare:nNnT \l_@@_columns_width_dim > \c_zero_dim
1998         {
1999             \skip_horizontal:N - \l_@@_columns_width_dim
2000             \bool_if:NTF \l_@@_tabular_bool
2001             { \skip_horizontal:n { - 2 \tabcolsep } }
2002             { \skip_horizontal:n { - 2 \arraycolsep } }
2003         }
2004     }
2005     \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.

```

2006 \bool_if:NT \l_@@_width_used_bool
2007 {

```

```

2008     \int_if_zero:nT \g_@@_total_X_weight_int
2009     { \@@_error_or_warning:n { width-without-X-columns } }
2010 }

```

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 .

```

2011     \int_compare:nNnT \g_@@_total_X_weight_int > \c_zero_int
2012     {
2013         \tl_gput_right:Nx \g_@@_aux_tl
2014         {
2015             \bool_set_true:N \l_@@_X_columns_aux_bool
2016             \dim_set:Nn \l_@@_X_columns_dim
2017             {
2018                 \dim_compare:nNnTF
2019                 {
2020                     \dim_abs:n
2021                     { \l_@@_width_dim - \box_wd:N \l_@@_the_array_box }
2022                 }
2023                 <
2024                 { 0.001 pt }
2025                 { \dim_use:N \l_@@_X_columns_dim }
2026                 {
2027                     \dim_eval:n
2028                     {
2029                         ( \l_@@_width_dim - \box_wd:N \l_@@_the_array_box )
2030                         / \int_use:N \g_@@_total_X_weight_int
2031                         + \l_@@_X_columns_dim
2032                     }
2033                 }
2034             }
2035         }
2036     }

```

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).

```

2037     \int_compare:nNnT \l_@@_last_row_int > { -2 }
2038     {
2039         \bool_if:NF \l_@@_last_row_without_value_bool
2040         {
2041             \int_compare:nNnF \l_@@_last_row_int = \c@iRow
2042             {
2043                 \@@_error:n { Wrong-last-row }
2044                 \int_gset_eq:NN \l_@@_last_row_int \c@iRow
2045             }
2046         }
2047     }

```

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

```

2048     \int_gset_eq:NN \c@jCol \g_@@_col_total_int
2049     \bool_if:NNT \g_@@_last_col_found_bool
2050     { \int_gdecr:N \c@jCol }
2051     {
2052         \int_compare:nNnT \l_@@_last_col_int > { -1 }
2053         { \@@_error:n { last-col-not-used } }
2054     }

```

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

```

2055     \int_gset_eq:NN \g_@@_row_total_int \c@iRow

```

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


```
2056 \int_compare:nNtT \l_@@_last_row_int > { -1 } { \int_gdecr:N \c@iRow }
```

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

```
2057 \int_if_zero:nT \l_@@_first_col_int
2058 { \skip_horizontal:N \g_@@_width_first_col_dim }
```

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

```
2059 \bool_if:nTF { ! \g_@@_delims_bool }
2060 {
2061   \tl_if_eq:NNTF \l_@@_baseline_tl \c_@@_c_tl
2062   \@@_use_arraybox_with_notes_c:
2063   {
2064     \tl_if_eq:NNTF \l_@@_baseline_tl \c_@@_b_tl
2065     \@@_use_arraybox_with_notes_b:
2066     \@@_use_arraybox_with_notes:
2067   }
2068 }
```

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).

```
2069 {
2070   \int_if_zero:nTF \l_@@_first_row_int
2071   {
2072     \dim_set_eq:NN \l_tmpa_dim \g_@@_dp_row_zero_dim
2073     \dim_add:Nn \l_tmpa_dim \g_@@_ht_row_zero_dim
2074   }
2075   { \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

```
2076 \int_compare:nNtTF \l_@@_last_row_int > { -2 }
2077 {
2078   \dim_set_eq:NN \l_tmpb_dim \g_@@_ht_last_row_dim
2079   \dim_add:Nn \l_tmpb_dim \g_@@_dp_last_row_dim
2080 }
2081 { \dim_zero:N \l_tmpb_dim }
2082 \hbox_set:Nn \l_tmpa_box
2083 {
2084   \c_math_toggle_token
2085   \@@_color:o \l_@@_delimiters_color_tl
2086   \exp_after:wN \left \g_@@_left_delim_tl
2087   \vcenter
2088   {
```

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.

```
2089   \skip_vertical:n { -\l_tmpa_dim - \arrayrulewidth }
2090   \hbox
2091   {
2092     \bool_if:NtF \l_@@_tabular_bool
2093     { \skip_horizontal:N -\tabcolsep }
2094     { \skip_horizontal:N -\arraycolsep }
2095     \@@_use_arraybox_with_notes_c:
2096     \bool_if:NtF \l_@@_tabular_bool
2097     { \skip_horizontal:N -\tabcolsep }
2098     { \skip_horizontal:N -\arraycolsep }
2099   }
```

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

```
2100   \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).

```

2101         \skip_vertical:N \arrayrulewidth
2102     }
2103     \exp_after:wN \right \g_@@_right_delim_tl
2104     \c_math_toggle_token
2105 }

```

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.

```

2106     \bool_if:NTF \l_@@_delimiters_max_width_bool
2107     {
2108         \@@_put_box_in_flow_bis:nn
2109         \g_@@_left_delim_tl
2110         \g_@@_right_delim_tl
2111     }
2112     \@@_put_box_in_flow:
2113 }

```

We take into account a potential “last column” (this “last column” has been constructed in an overlapping position and we have computed its width in `\g_@@_width_last_col_dim`: see p. 90).

```

2114     \bool_if:NT \g_@@_last_col_found_bool
2115     { \skip_horizontal:N \g_@@_width_last_col_dim }
2116     \bool_if:NT \l_@@_preamble_bool
2117     {
2118         \int_compare:nNnT \c@jCol < \g_@@_static_num_of_col_int
2119         { \@@_warning_gredirect_none:n { columns-not-used } }
2120     }
2121     \@@_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.

```

2122     \egroup

```

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

```

2123     \iow_now:Nn \@mainaux { \ExplSyntaxOn }
2124     \iow_now:Nn \@mainaux { \char_set_catcode_space:n { 32 } }
2125     \iow_now:Nx \@mainaux
2126     {
2127         \tl_gset:cn { c_@@_ \int_use:N \g_@@_env_int _ tl }
2128         { \exp_not:o \g_@@_aux_tl }
2129     }
2130     \iow_now:Nn \@mainaux { \ExplSyntaxOff }

2131     \bool_if:NT \g_@@_footnote_bool \endsavenotes
2132 }

```

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.

```

2133 \cs_new_protected:Npn \@@_transform_preamble:
2134 {
2135     \@@_transform_preamble_i:
2136     \@@_transform_preamble_ii:
2137 }

```

```

2138 \cs_new_protected:Npn \@@_transform_preamble_i:
2139 {
2140   \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`).

```

2141   \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.

```

2142   \bool_gset_false:N \g_tmpb_bool

```

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

```

2143   \tl_gclear_new:N \g_@@_pre_cell_tl

```

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

```

2144   \int_zero:N \l_tmpa_int
2145   \tl_gclear:N \g_@@_array_preamble_tl
2146   \tl_if_eq:NNTF \l_@@_vlines_clist \c_@@_all_tl
2147   {
2148     \tl_gset:Nn \g_@@_array_preamble_tl
2149       { ! { \skip_horizontal:N \arrayrulewidth } }
2150   }
2151   {
2152     \clist_if_in:NnT \l_@@_vlines_clist 1
2153     {
2154       \tl_gset:Nn \g_@@_array_preamble_tl
2155         { ! { \skip_horizontal:N \arrayrulewidth } }
2156     }
2157   }

```

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

```

2158   \exp_last_unbraced:NV \@@_rec_preamble:n \g_@@_user_preamble_tl \stop
2159   \int_gset_eq:NN \g_@@_static_num_of_col_int \c@jCol

2160   \@@_replace_columncolor:
2161 }

```

```

2162 \hook_gput_code:nnn { begindocument } { . }
2163 {
2164   \IfPackageLoadedTF { colortbl }
2165   {

```

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).

```

2166     \regex_const:Nn \c_@@_columncolor_regex { \c { columncolor } }
2167     \cs_new_protected:Npn \@@_replace_columncolor:
2168     {
2169       \regex_replace_all:NnN
2170         \c_@@_columncolor_regex
2171         { \c { @@_columncolor_preamble } }
2172       \g_@@_array_preamble_tl
2173     }
2174   }
2175   {
2176     \cs_new_protected:Npn \@@_replace_columncolor:
2177     { \cs_set_eq:NN \columncolor \@@_columncolor_preamble }
2178   }
2179 }

```

```

2180 \cs_new_protected:Npn \@@_transform_preamble_ii:
2181 {

```

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

```

2182 \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2183 {
2184   \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2185   { \bool_gset_true:N \g_@@_delims_bool }
2186 }
2187 { \bool_gset_true:N \g_@@_delims_bool }

```

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

```

2188 \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).

```

2189 \int_if_zero:nTF \l_@@_first_col_int
2190 { \tl_gput_left:No \g_@@_array_preamble_tl \c_@@_preamble_first_col_tl }
2191 {
2192   \bool_if:NF \g_@@_delims_bool
2193   {
2194     \bool_if:NF \l_@@_tabular_bool
2195     {
2196       \tl_if_empty:NT \l_@@_vlines_clist
2197       {
2198         \bool_if:NF \l_@@_exterior_arraycolsep_bool
2199         { \tl_gput_left:Nn \g_@@_array_preamble_tl { @ { } } }
2200       }
2201     }
2202   }
2203 }
2204 \int_compare:nNnTF \l_@@_last_col_int > { -1 }
2205 { \tl_gput_right:No \g_@@_array_preamble_tl \c_@@_preamble_last_col_tl }
2206 {
2207   \bool_if:NF \g_@@_delims_bool
2208   {
2209     \bool_if:NF \l_@@_tabular_bool
2210     {
2211       \tl_if_empty:NT \l_@@_vlines_clist
2212       {
2213         \bool_if:NF \l_@@_exterior_arraycolsep_bool
2214         { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { } } }
2215       }
2216     }
2217   }
2218 }

```

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`).

```

2219 \dim_compare:nNnT \l_@@_tabular_width_dim = \c_zero_dim
2220 {
2221   \tl_gput_right:Nn \g_@@_array_preamble_tl
2222   { > { \@@_error_too_much_cols: } 1 }
2223 }
2224 }

```

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.

```

2225 \cs_new_protected:Npn \@@_rec_preamble:n #1
2226 {

```

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

```
2227 \cs_if_exist:cTF { @@ _ \token_to_str:N #1 }
2228 { \use:c { @@ _ \token_to_str:N #1 } { #1 } }
2229 {
```

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

```
2230 \cs_if_exist:cTF { NC @ find @ #1 }
2231 {
2232 \tl_set_eq:Nc \l_tmpb_tl { NC @ rewrite @ #1 }
2233 \exp_last_unbraced:N \@@_rec_preamble:n \l_tmpb_tl
2234 }
2235 {
2236 \tl_if_eq:nnT { #1 } { S }
2237 { \@@_fatal:n { unknown~column~type~S } }
2238 { \@@_fatal:nn { unknown~column~type } { #1 } }
2239 }
2240 }
2241 }
```

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

```
2242 \cs_new:Npn \@@_c #1
2243 {
2244 \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2245 \tl_gclear:N \g_@@_pre_cell_tl
2246 \tl_gput_right:Nn \g_@@_array_preamble_tl
2247 { > \@@_cell_begin:w c < \@@_cell_end: }
```

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

```
2248 \int_gincr:N \c@jCol
2249 \@@_rec_preamble_after_col:n
2250 }

2251 \cs_new:Npn \@@_l #1
2252 {
2253 \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2254 \tl_gclear:N \g_@@_pre_cell_tl
2255 \tl_gput_right:Nn \g_@@_array_preamble_tl
2256 {
2257 > { \@@_cell_begin:w \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_l_tl }
2258 l
2259 < \@@_cell_end:
2260 }
2261 \int_gincr:N \c@jCol
2262 \@@_rec_preamble_after_col:n
2263 }

2264 \cs_new:Npn \@@_r #1
2265 {
2266 \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2267 \tl_gclear:N \g_@@_pre_cell_tl
2268 \tl_gput_right:Nn \g_@@_array_preamble_tl
2269 {
2270 > { \@@_cell_begin:w \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_r_tl }
2271 r
2272 < \@@_cell_end:
2273 }
2274 \int_gincr:N \c@jCol
2275 \@@_rec_preamble_after_col:n
2276 }
```

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 @

```

2277 \cs_new:cpn { @@ _ \token_to_str:N ! } #1 #2
2278 {
2279   \tl_gput_right:Nn \g_@@_array_preamble_tl { #1 { #2 } }
2280   \@@_rec_preamble:n
2281 }
2282 \cs_set_eq:cc { @@ _ \token_to_str:N @ } { @@ _ \token_to_str:N ! }

```

For |

```

2283 \cs_new:cpn { @@ _ | } #1
2284 {

```

\l_tmpa_int is the number of successive occurrences of |

```

2285   \int_incr:N \l_tmpa_int
2286   \@@_make_preamble_i_i:n
2287 }

2288 \cs_new_protected:Npn \@@_make_preamble_i_i:n #1
2289 {
2290   \str_if_eq:nnTF { #1 } |
2291   { \use:c { @@ _ | } | }
2292   { \@@_make_preamble_i_ii:nn { } #1 }
2293 }

2294 \cs_new_protected:Npn \@@_make_preamble_i_ii:nn #1 #2
2295 {
2296   \str_if_eq:nnTF { #2 } [
2297   { \@@_make_preamble_i_iii:nn { #1 } [ ] }
2298   { \@@_make_preamble_i_iii:nn { #2 } { #1 } }
2299 ]

2300 \cs_new_protected:Npn \@@_make_preamble_i_ii:nw #1 [ #2 ]
2301 { \@@_make_preamble_i_ii:nn { #1 , #2 } }

2302 \cs_new_protected:Npn \@@_make_preamble_i_iii:nn #1 #2
2303 {
2304   \@@_compute_rule_width:n { multiplicity = \l_tmpa_int , #2 }
2305   \tl_gput_right:Nx \g_@@_array_preamble_tl
2306   {

```

Here, the command \dim_eval:n is mandatory.

```

2307   \exp_not:N ! { \skip_horizontal:n { \dim_eval:n { \l_@@_rule_width_dim } } }
2308 }
2309 \tl_gput_right:Nx \g_@@_pre_code_after_tl
2310 {
2311   \@@_vline:n
2312   {
2313     position = \int_eval:n { \c@jCol + 1 } ,
2314     multiplicity = \int_use:N \l_tmpa_int ,
2315     total-width = \dim_use:N \l_@@_rule_width_dim ,
2316     #2
2317   }

```

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.

```

2318   }
2319   \int_zero:N \l_tmpa_int
2320   \str_if_eq:nnT { #1 } { \stop } { \bool_gset_true:N \g_tmpb_bool }
2321   \@@_rec_preamble:n #1
2322 }

2323 \cs_new:cpn { @@ _ > } #1 #2
2324 {
2325   \tl_gput_right:Nn \g_@@_pre_cell_tl { > { #2 } }
2326   \@@_rec_preamble:n
2327 }

```

```
2328 \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.

```
2329 \keys_define:nn { nicematrix / p-column }
2330 {
2331   r .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_r_str ,
2332   r .value_forbidden:n = true ,
2333   c .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_c_str ,
2334   c .value_forbidden:n = true ,
2335   l .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_l_str ,
2336   l .value_forbidden:n = true ,
2337   R .code:n =
2338     \IfPackageLoadedTF { ragged2e }
2339     { \str_set_eq:NN \l_@@_hpos_col_str \c_@@_R_str }
2340     {
2341       \@@_error_or_warning:n { ragged2e~not~loaded }
2342       \str_set_eq:NN \l_@@_hpos_col_str \c_@@_r_str
2343     } ,
2344   R .value_forbidden:n = true ,
2345   L .code:n =
2346     \IfPackageLoadedTF { ragged2e }
2347     { \str_set_eq:NN \l_@@_hpos_col_str \c_@@_L_str }
2348     {
2349       \@@_error_or_warning:n { ragged2e~not~loaded }
2350       \str_set_eq:NN \l_@@_hpos_col_str \c_@@_l_str
2351     } ,
2352   L .value_forbidden:n = true ,
2353   C .code:n =
2354     \IfPackageLoadedTF { ragged2e }
2355     { \str_set_eq:NN \l_@@_hpos_col_str \c_@@_C_str }
2356     {
2357       \@@_error_or_warning:n { ragged2e~not~loaded }
2358       \str_set_eq:NN \l_@@_hpos_col_str \c_@@_c_str
2359     } ,
2360   C .value_forbidden:n = true ,
2361   S .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_si_str ,
2362   S .value_forbidden:n = true ,
2363   p .code:n = \str_set:Nn \l_@@_vpos_col_str { p } ,
2364   p .value_forbidden:n = true ,
2365   t .meta:n = p ,
2366   m .code:n = \str_set:Nn \l_@@_vpos_col_str { m } ,
2367   m .value_forbidden:n = true ,
2368   b .code:n = \str_set:Nn \l_@@_vpos_col_str { b } ,
2369   b .value_forbidden:n = true ,
2370 }
```

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

```
2371 \cs_new:Npn \@@_p #1
2372 {
2373   \str_set:Nn \l_@@_vpos_col_str { #1 }
```

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

```
2374   \@@_make_preamble_ii_i:n
2375 }
2376 \cs_set_eq:NN \@@_b \@@_p
2377 \cs_set_eq:NN \@@_m \@@_p
2378 \cs_new_protected:Npn \@@_make_preamble_ii_i:n #1
2379 {
2380   \str_if_eq:nnTF { #1 } { [ ]
2381     { \@@_make_preamble_ii_ii:w [ ]
2382       { \@@_make_preamble_ii_ii:w [ ] { #1 } }
2383     }
```

```

2384 \cs_new_protected:Npn \@@_make_preamble_ii_ii:w [ #1 ]
2385 { \@@_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.

```

2386 \cs_new_protected:Npn \@@_make_preamble_ii_iii:nn #1 #2
2387 {

```

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).

```

2388   \str_set_eq:NN \l_@@_hpos_col_str \c_@@_j_str
2389   \@@_keys_p_column:n { #1 }
2390   \@@_make_preamble_ii_iv:nnn { #2 } { minipage } { }
2391 }

2392 \cs_new_protected:Npn \@@_keys_p_column:n #1
2393 { \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.

```

2394 \cs_new_protected:Npn \@@_make_preamble_ii_iv:nnn #1 #2 #3
2395 {
2396   \use:e
2397   {
2398     \@@_make_preamble_ii_v:nnnnnnnn
2399     { \str_if_eq:onTF \l_@@_vpos_col_str { p } { t } { b } }
2400     { \dim_eval:n { #1 } }
2401   }

```

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.

```

2402   \str_if_eq:NNTF \l_@@_hpos_col_str \c_@@_j_str
2403   { \tl_clear:N \exp_not:N \l_@@_hpos_cell_tl }
2404   {
2405     \cs_set_nopar:Npn \exp_not:N \l_@@_hpos_cell_tl
2406     { \str_lowercase:o \l_@@_hpos_col_str }
2407   }
2408   \str_case:on \l_@@_hpos_col_str
2409   {
2410     c { \exp_not:N \centering }
2411     l { \exp_not:N \raggedright }
2412     r { \exp_not:N \raggedleft }
2413     C { \exp_not:N \Centering }
2414     L { \exp_not:N \RaggedRight }
2415     R { \exp_not:N \RaggedLeft }
2416   }
2417   #3
2418 }
2419 { \str_if_eq:onT \l_@@_vpos_col_str { m } \@@_center_cell_box: }
2420 { \str_if_eq:onT \l_@@_hpos_col_str { si } \siunitx_cell_begin:w }
2421 { \str_if_eq:onT \l_@@_hpos_col_str { si } \siunitx_cell_end: }
2422 { #2 }
2423 {
2424   \str_case:onF \l_@@_hpos_col_str
2425   {
2426     { j } { c }
2427     { si } { c }
2428   }

```

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

```

2429     { \str_lowercase:o \l_@@_hpos_col_str }
2430   }
2431 }

```


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

```

2432     \int_gincr:N \c@jCol
2433     \@@_rec_preamble_after_col:n
2434 }

```

#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*.

```

2435 \cs_new_protected:Npn \@@_make_preamble_ii_v:nnnnnnnn #1 #2 #3 #4 #5 #6 #7 #8
2436 {
2437     \tl_if_eq:NNTF \l_@@_hpos_col_str \c_@@_si_str
2438     { \tl_gput_right:Nn \g_@@_array_preamble_tl { > { \@@_test_if_empty_for_S: } } }
2439     { \tl_gput_right:Nn \g_@@_array_preamble_tl { > { \@@_test_if_empty: } } }
2440     \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2441     \tl_gclear:N \g_@@_pre_cell_tl
2442     \tl_gput_right:Nn \g_@@_array_preamble_tl
2443     {
2444         > {

```

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.

```

2445         \dim_set:Nn \l_@@_col_width_dim { #2 }
2446         \bool_if:NT \c_@@_testphase_table_bool
2447         { \tag_struct_begin:n { tag = Div } }
2448         \@@_cell_begin:w

```

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

```

2449         \use:c { #7 } [ #1 ] { #2 }

```

The following lines have been taken from array.sty.

```

2450     \everypar
2451     {
2452         \vrule height \box_ht:N \@arstrutbox width \c_zero_dim
2453         \everypar { }
2454     }
2455     \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.).

```

2456     #3

```

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

```

2457     \g_@@_row_style_tl
2458     \arraybackslash
2459     #5
2460 }
2461 #8
2462 < {
2463     #6

```

The following line has been taken from array.sty.

```

2464     \@finalstrut \@arstrutbox
2465     \use:c { end #7 }

```

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

```

2466         #4
2467         \@@_cell_end:
2468         \bool_if:NT \c_@@_testphase_table_bool \tag_struct_end:
2469     }
2470 }
2471 }

2472 \str_new:N \c_@@_ignorespaces_str
2473 \str_set:Nx \c_@@_ignorespaces_str { \ignorespaces }
2474 \str_remove_all:Nn \c_@@_ignorespaces_str { ~ }
2475 \cs_new_protected:Npn \@@_test_if_empty:
2476 { \peek_after:Nw \@@_test_if_empty_i: }
2477 \cs_new_protected:Npn \@@_test_if_empty_i:
2478 {
2479     \str_set:Nx \l_tmpa_str { \token_to_meaning:N \l_peek_token }
2480     \str_if_eq:NNT \l_tmpa_str \c_@@_ignorespaces_str
2481     { \@@_test_if_empty:w }
2482 }
2483 \cs_new_protected:Npn \@@_test_if_empty:w \ignorespaces
2484 { \peek_after:Nw \@@_test_if_empty_ii: }

2485 \cs_new_protected:Npn \@@_nullify_cell:
2486 {
2487     \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2488     {
2489         \box_set_wd:Nn \l_@@_cell_box \c_zero_dim
2490         \skip_horizontal:N \l_@@_col_width_dim
2491     }
2492 }

2493 \bool_if:NTF \c_@@_tagging_array_bool
2494 {
2495     \cs_new_protected:Npn \@@_test_if_empty_ii:
2496     { \peek_meaning:NT \textonly@unskip \@@_nullify_cell: }
2497 }

```

In the old version of array, we test whether it begins by \ignorespaces\unskip. However, in some circumstances, for example when \collectcell of collcell 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...

```

2498 {
2499     \cs_new_protected:Npn \@@_test_if_empty_ii:
2500     { \peek_meaning:NT \unskip \@@_nullify_cell: }
2501 }

2502 \cs_new_protected:Npn \@@_test_if_empty_for_S:
2503 {
2504     \peek_meaning:NT \__siunitx_table_skip:n
2505     {
2506         \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2507         { \box_set_wd:Nn \l_@@_cell_box \c_zero_dim }
2508     }
2509 }

```

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

```

2510 \cs_new_protected:Npn \@@_center_cell_box:
2511 {

```

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

```

2512 \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2513 {
2514   \int_compare:nNnT
2515     { \box_ht:N \l_@@_cell_box }
2516     >

```

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).

```

2517   { \box_ht:N \strutbox }
2518   {
2519     \hbox_set:Nn \l_@@_cell_box
2520     {
2521       \box_move_down:nn
2522       {
2523         ( \box_ht:N \l_@@_cell_box - \box_ht:N \@arstrutbox
2524           + \baselineskip ) / 2
2525       }
2526       { \box_use:N \l_@@_cell_box }
2527     }
2528   }
2529 }
2530 }

```

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

```

2531 \cs_new:Npn \@@_V #1 #2
2532 {
2533   \str_if_eq:nnTF { #2 } { [ ] }
2534     { \@@_make_preamble_V_i:w [ ] }
2535     { \@@_make_preamble_V_i:w [ ] { #2 } }
2536 }
2537 \cs_new_protected:Npn \@@_make_preamble_V_i:w [ #1 ]
2538 { \@@_make_preamble_V_ii:nn { #1 } }
2539 \cs_new_protected:Npn \@@_make_preamble_V_ii:nn #1 #2
2540 {
2541   \str_set:Nn \l_@@_vpos_col_str { p }
2542   \str_set_eq:NN \l_@@_hpos_col_str \c_@@_j_str
2543   \@@_keys_p_column:n { #1 }
2544   \IfPackageLoadedTF { varwidth }
2545     { \@@_make_preamble_ii_iv:nnn { #2 } { varwidth } { } }
2546     {
2547       \@@_error_or_warning:n { varwidth-not-loaded }
2548       \@@_make_preamble_ii_iv:nnn { #2 } { minipage } { }
2549     }
2550 }

```

For `w` and `W`

```

2551 \cs_new:Npn \@@_w { \@@_make_preamble_w:nnnn { } }
2552 \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.

```

2553 \cs_new_protected:Npn \@@_make_preamble_w:nnnn #1 #2 #3 #4
2554 {
2555   \str_if_eq:nnTF { #3 } { s }
2556     { \@@_make_preamble_w_i:nnnn { #1 } { #4 } }
2557     { \@@_make_preamble_w_ii:nnnn { #1 } { #2 } { #3 } { #4 } }
2558 }

```

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.

```

2559 \cs_new_protected:Npn \@@_make_preamble_w_i:nnnn #1 #2
2560 {
2561   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2562   \tl_gclear:N \g_@@_pre_cell_tl
2563   \tl_gput_right:Nn \g_@@_array_preamble_tl
2564     {
2565       > {
2566         \dim_set:Nn \l_@@_col_width_dim { #2 }
2567         \@@_cell_begin:w
2568         \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_c_tl
2569       }
2570       c
2571       < {
2572         \@@_cell_end_for_w_s:
2573         #1
2574         \@@_adjust_size_box:
2575         \box_use_drop:N \l_@@_cell_box
2576       }
2577     }
2578   \int_gincr:N \c@jCol
2579   \@@_rec_preamble_after_col:n
2580 }

```

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

```

2581 \cs_new_protected:Npn \@@_make_preamble_w_ii:nnnn #1 #2 #3 #4
2582 {
2583   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2584   \tl_gclear:N \g_@@_pre_cell_tl
2585   \tl_gput_right:Nn \g_@@_array_preamble_tl
2586     {
2587       > {

```

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.

```

2588         \dim_set:Nn \l_@@_col_width_dim { #4 }
2589         \hbox_set:Nw \l_@@_cell_box
2590         \@@_cell_begin:w
2591         \cs_set_nopar:Npn \l_@@_hpos_cell_tl { #3 }
2592       }
2593       c
2594       < {
2595         \@@_cell_end:
2596         \hbox_set_end:
2597         #1
2598         \@@_adjust_size_box:
2599         \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2600       }
2601     }

```

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

```

2602   \int_gincr:N \c@jCol
2603   \@@_rec_preamble_after_col:n
2604 }

```

```

2605 \cs_new_protected:Npn \@@_special_W:
2606 {
2607   \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \l_@@_col_width_dim
2608     { \@@_warning:n { W~warning } }
2609 }

```

For S (of siunitx).

```

2610 \cs_new:Npn \@@_S #1 #2
2611 {
2612   \str_if_eq:nnTF { #2 } { [ ] }
2613   { \@@_make_preamble_S:w [ ] }
2614   { \@@_make_preamble_S:w [ ] { #2 } }
2615 }
2616 \cs_new_protected:Npn \@@_make_preamble_S:w [ #1 ]
2617 { \@@_make_preamble_S_i:n { #1 } }
2618 \cs_new_protected:Npn \@@_make_preamble_S_i:n #1
2619 {
2620   \IfPackageLoadedTF { siunitx }
2621   {
2622     \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2623     \tl_gclear:N \g_@@_pre_cell_tl
2624     \tl_gput_right:Nn \g_@@_array_preamble_tl
2625     {
2626       > {
2627         \@@_cell_begin:w
2628         \keys_set:nn { siunitx } { #1 }
2629         \siunitx_cell_begin:w
2630       }
2631       c
2632       < { \siunitx_cell_end: \@@_cell_end: }
2633     }

```

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

```

2634   \int_gincr:N \c@jCol
2635   \@@_rec_preamble_after_col:n
2636 }
2637 { \@@_fatal:n { siunitx~not~loaded } }
2638 }

```

For (, [and \{.

```

2639 \cs_new:cpn { @@ _ \token_to_str:N ( } #1 #2
2640 {
2641   \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.

```

2642   \int_if_zero:nTF \c@jCol
2643   {
2644     \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2645     {

```

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

```

2646     \tl_gset:Nn \g_@@_left_delim_tl { #1 }
2647     \tl_gset_eq:NN \g_@@_right_delim_tl \c_@@_dot_tl
2648     \@@_rec_preamble:n #2
2649   }
2650   {
2651     \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2652     \@@_make_preamble_iv:nn { #1 } { #2 }
2653   }
2654 }
2655 { \@@_make_preamble_iv:nn { #1 } { #2 } }
2656 }
2657 \cs_set_eq:cc { @@ _ \token_to_str:N [ ] { @@ _ \token_to_str:N ( }
2658 \cs_set_eq:cc { @@ _ \token_to_str:N \{ } { @@ _ \token_to_str:N ( }
2659 \cs_new_protected:Npn \@@_make_preamble_iv:nn #1 #2
2660 {
2661   \tl_gput_right:Nx \g_@@_pre_code_after_tl
2662   { \@@_delimiter:nnn #1 { \int_eval:n { \c@jCol + 1 } } } \c_true_bool }

```

```

2663 \tl_if_in:nnTF { ( [ \{ ) ] \} \left \right } { #2 }
2664 {
2665   \@@_error:nn { delimiter~after~opening } { #2 }
2666   \@@_rec_preamble:n
2667 }
2668 { \@@_rec_preamble:n #2 }
2669 }

```

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

```

2670 \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}`).

```

2671 \cs_new:cpn { @@ _ \token_to_str:N ) } #1 #2
2672 {
2673   \bool_if:NT \l_@@_small_bool { \@@_fatal:n { Delimiter~with~small } }
2674   \tl_if_in:nnTF { ) ] \} } { #2 }
2675   { \@@_make_preamble_v:nnn #1 #2 }
2676   {
2677     \tl_if_eq:nnTF { \stop } { #2 }
2678     {
2679       \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2680       { \tl_gset:Nn \g_@@_right_delim_tl { #1 } }
2681       {
2682         \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2683         \tl_gput_right:Nx \g_@@_pre_code_after_tl
2684         { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2685         \@@_rec_preamble:n #2
2686       }
2687     }
2688     {
2689       \tl_if_in:nnT { ( [ \{ \left } { #2 }
2690       { \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } } }
2691       \tl_gput_right:Nx \g_@@_pre_code_after_tl
2692       { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2693       \@@_rec_preamble:n #2
2694     }
2695   }
2696 }
2697 \cs_set_eq:cc { @@ _ \token_to_str:N ] } { @@ _ \token_to_str:N ) }
2698 \cs_set_eq:cc { @@ _ \token_to_str:N \} } { @@ _ \token_to_str:N ) }
2699 \cs_new_protected:Npn \@@_make_preamble_v:nnn #1 #2 #3
2700 {
2701   \tl_if_eq:nnTF { \stop } { #3 }
2702   {
2703     \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2704     {
2705       \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2706       \tl_gput_right:Nx \g_@@_pre_code_after_tl
2707       { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2708       \tl_gset:Nn \g_@@_right_delim_tl { #2 }
2709     }
2710     {
2711       \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2712       \tl_gput_right:Nx \g_@@_pre_code_after_tl
2713       { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2714       \@@_error:nn { double~closing~delimiter } { #2 }
2715     }
2716   }
2717   {
2718     \tl_gput_right:Nx \g_@@_pre_code_after_tl

```

```

2719         { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2720         \@@_error:nn { double~closing~delimiter } { #2 }
2721         \@@_rec_preamble:n #3
2722     }
2723 }

```

```

2724 \cs_new:cpn { @@ _ \token_to_str:N \right } #1
2725 { \use:c { @@ _ \token_to_str:N ) } }

```

After a specifier of column, we have to test whether there is one or several $\langle \dots \rangle$ because, after those potential $\langle \dots \rangle$, 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 $\langle \dots \rangle$, a $\@{\dots}$.

```

2726 \cs_new_protected:Npn \@@_rec_preamble_after_col:n #1
2727 {
2728     \str_if_eq:nnTF { #1 } { < }
2729     \@@_rec_preamble_after_col_i:n
2730     {
2731         \str_if_eq:nnTF { #1 } { @ }
2732         \@@_rec_preamble_after_col_ii:n
2733         {
2734             \tl_if_eq:NNTF \l_@@_vlines_clist \c_@@_all_tl
2735             {
2736                 \tl_gput_right:Nn \g_@@_array_preamble_tl
2737                 { ! { \skip_horizontal:N \arrayrulewidth } }
2738             }
2739             {
2740                 \exp_args:NNe
2741                 \clist_if_in:NnT \l_@@_vlines_clist { \int_eval:n { \c@jCol + 1 } }
2742                 {
2743                     \tl_gput_right:Nn \g_@@_array_preamble_tl
2744                     { ! { \skip_horizontal:N \arrayrulewidth } }
2745                 }
2746             }
2747             \@@_rec_preamble:n { #1 }
2748         }
2749     }
2750 }
2751 \cs_new_protected:Npn \@@_rec_preamble_after_col_i:n #1
2752 {
2753     \tl_gput_right:Nn \g_@@_array_preamble_tl { < { #1 } }
2754     \@@_rec_preamble_after_col:n
2755 }

```

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.

```

2756 \cs_new_protected:Npn \@@_rec_preamble_after_col_ii:n #1
2757 {
2758     \tl_if_eq:NNTF \l_@@_vlines_clist \c_@@_all_tl
2759     {
2760         \tl_gput_right:Nn \g_@@_array_preamble_tl
2761         { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2762     }
2763     {
2764         \exp_args:NNe
2765         \clist_if_in:NnTF \l_@@_vlines_clist { \int_eval:n { \c@jCol + 1 } }
2766         {
2767             \tl_gput_right:Nn \g_@@_array_preamble_tl
2768             { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2769         }
2770         { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { #1 } } }
2771     }
2772     \@@_rec_preamble:n
2773 }

```

```

2774 \cs_new:cpn { @@ _ * } #1 #2 #3
2775 {
2776   \tl_clear:N \l_tmpa_tl
2777   \int_step_inline:nn { #2 } { \tl_put_right:Nn \l_tmpa_tl { #3 } }
2778   \exp_last_unbraced:No \@@_rec_preamble:n \l_tmpa_tl
2779 }

```

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.

```

2780 \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.

```

2781 \cs_new:Npn \@@_X #1 #2
2782 {
2783   \str_if_eq:nnTF { #2 } { [ ]
2784     { \@@_make_preamble_X:w [ ] }
2785     { \@@_make_preamble_X:w [ ] #2 }
2786   }
2787   \cs_new_protected:Npn \@@_make_preamble_X:w [ #1 ]
2788   { \@@_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`).

```

2789 \keys_define:nn { nicematrix / X-column }
2790 { 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.

```

2791 \cs_new_protected:Npn \@@_make_preamble_X_i:n #1
2792 {

```

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).

```

2793   \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).

```

2794   \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`.

```

2795   \int_zero_new:N \l_@@_weight_int
2796   \int_set_eq:NN \l_@@_weight_int \c_one_int
2797   \@@_keys_p_column:n { #1 }
2798   \keys_set:no { nicematrix / X-column } \l_tmpa_tl
2799   \int_compare:nNnT \l_@@_weight_int < \c_zero_int
2800   {
2801     \@@_error_or_warning:n { negative~weight }
2802     \int_set:Nn \l_@@_weight_int { - \l_@@_weight_int }
2803   }
2804   \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).

```

2805 \bool_if:NTF \l_@@_X_columns_aux_bool
2806 {
2807   \exp_args:Nne
2808   \@@_make_preamble_ii_iv:nnn
2809   { \l_@@_weight_int \l_@@_X_columns_dim }
2810   { minipage }
2811   { \@@_no_update_width: }
2812 }
2813 {
2814   \tl_gput_right:Nn \g_@@_array_preamble_tl
2815   {
2816     > {
2817       \@@_cell_begin:w
2818       \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`.

```

2819 \NotEmpty

```

The following code will nullify the box of the cell.

```

2820 \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2821 { \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`.

```

2822 \begin { minipage } { 5 cm } \arraybackslash
2823 }
2824 c
2825 < {
2826   \end { minipage }
2827   \@@_cell_end:
2828 }
2829 }
2830 \int_gincr:N \c@jCol
2831 \@@_rec_preamble_after_col:n
2832 }
2833 }

```

```

2834 \cs_new_protected:Npn \@@_no_update_width:
2835 {
2836   \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2837   { \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing: }
2838 }

```

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

```

2839 \cs_new_protected:Npn \@@_make_preamble_vlism:n #1
2840 {
2841   \seq_gput_right:Nx \g_@@_cols_vlism_seq
2842   { \int_eval:n { \c@jCol + 1 } }
2843   \tl_gput_right:Nx \g_@@_array_preamble_tl
2844   { \exp_not:N ! { \skip_horizontal:N \arrayrulewidth } }
2845   \@@_rec_preamble:n
2846 }

```

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.

```

2847 \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).

```

2848 \cs_new_protected:cpn { @@ _ \token_to_str:N \hline }
2849 { \@@_fatal:n { Preamble-forgotten } }
2850 \cs_set_eq:cc { @@ _ \token_to_str:N \Hline } { @@ _ \token_to_str:N \hline }
2851 \cs_set_eq:cc { @@ _ \token_to_str:N \toprule } { @@ _ \token_to_str:N \hline }
2852 \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).

```

2853 \cs_new:Npn \@@_multicolumn:nnn #1 #2 #3
2854 {

```

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`.

```

2855 \multispan { #1 }
2856 \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing:
2857 \begingroup
2858 \bool_if:NT \c_@@_testphase_table_bool
2859 { \tbl_update_multicolumn_cell_data:n { #1 } }
2860 \cs_set_nopar:Npn \@addamp
2861 { \legacy_if:nTF { @firstamp } { \@firstampfalse } { \@preamerr 5 } }

```

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

```

2862 \tl_gclear:N \g_@@_preamble_tl
2863 \@@_make_m_preamble:n #2 \q_stop

```

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

```

2864 \exp_args:No \@mkpream \g_@@_preamble_tl
2865 \@addtopreamble \@empty
2866 \endgroup
2867 \bool_if:NT \c_@@_testphase_table_bool
2868 { \UseTaggingSocket { tbl / colspan } { #1 } }

```

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

```

2869 \int_compare:nNnT { #1 } > \c_one_int
2870 {
2871 \seq_gput_left:Nx \g_@@_multicolumn_cells_seq
2872 { \int_use:N \c@iRow - \int_eval:n { \c@jCol + 1 } }
2873 \seq_gput_left:Nn \g_@@_multicolumn_sizes_seq { #1 }
2874 \seq_gput_right:Nx \g_@@_pos_of_blocks_seq
2875 {
2876 {
2877 \int_if_zero:nTF \c@jCol
2878 { \int_eval:n { \c@iRow + 1 } }
2879 { \int_use:N \c@iRow }
2880 }
2881 { \int_eval:n { \c@jCol + 1 } }
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 { } % for the name of the block

```

```

2889     }
2890 }

```

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

```

2891 \cs_set_nopar:Npn \@sharp { #3 }
2892 \@arstrut
2893 \@preamble
2894 \null

```

We add some lines.

```

2895 \int_gadd:Nn \c@jCol { #1 - 1 }
2896 \int_compare:nNnT \c@jCol > \g_@@_col_total_int
2897 { \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
2898 \ignorespaces
2899 }

```

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`.

```

2900 \cs_new_protected:Npn \@@_make_m_preamble:n #1
2901 {
2902   \str_case:nnF { #1 }
2903   {
2904     c { \@@_make_m_preamble_i:n #1 }
2905     l { \@@_make_m_preamble_i:n #1 }
2906     r { \@@_make_m_preamble_i:n #1 }
2907     > { \@@_make_m_preamble_ii:nn #1 }
2908     ! { \@@_make_m_preamble_ii:nn #1 }
2909     @ { \@@_make_m_preamble_ii:nn #1 }
2910     | { \@@_make_m_preamble_iii:n #1 }
2911     p { \@@_make_m_preamble_iv:nnn t #1 }
2912     m { \@@_make_m_preamble_iv:nnn c #1 }
2913     b { \@@_make_m_preamble_iv:nnn b #1 }
2914     w { \@@_make_m_preamble_v:nnnn { } #1 }
2915     W { \@@_make_m_preamble_v:nnnn { \@@_special_W: } #1 }
2916     \q_stop { }
2917   }
2918   {
2919     \cs_if_exist:cTF { NC @ find @ #1 }
2920     {
2921       \tl_set_eq:Nc \l_tmpa_tl { NC @ rewrite @ #1 }
2922       \exp_last_unbraced:No \@@_make_m_preamble:n \l_tmpa_tl
2923     }
2924     {
2925       \tl_if_eq:nnT { #1 } { S }
2926       { \@@_fatal:n { unknown~column~type~S } }
2927       { \@@_fatal:nn { unknown~column~type } { #1 } }
2928     }
2929   }
2930 }

```

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

```

2931 \cs_new_protected:Npn \@@_make_m_preamble_i:n #1
2932 {
2933   \tl_gput_right:Nn \g_@@_preamble_tl
2934   {
2935     > { \@@_cell_begin:w \cs_set_nopar:Npn \l_@@_hpos_cell_tl { #1 } }
2936     #1
2937     < \@@_cell_end:
2938   }

```

We test for the presence of a `<`.

```

2939   \@@_make_m_preamble_x:n
2940 }

```

For >, ! and @

```

2941 \cs_new_protected:Npn \@@_make_m_preamble_ii:nn #1 #2
2942 {
2943   \tl_gput_right:Nn \g_@@_preamble_tl { #1 { #2 } }
2944   \@@_make_m_preamble:n
2945 }

```

For |

```

2946 \cs_new_protected:Npn \@@_make_m_preamble_iii:n #1
2947 {
2948   \tl_gput_right:Nn \g_@@_preamble_tl { #1 }
2949   \@@_make_m_preamble:n
2950 }

```

For p, m and b

```

2951 \cs_new_protected:Npn \@@_make_m_preamble_iv:nnn #1 #2 #3
2952 {
2953   \tl_gput_right:Nn \g_@@_preamble_tl
2954   {
2955     > {
2956       \@@_cell_begin:w
2957       \begin { minipage } [ #1 ] { \dim_eval:n { #3 } }
2958       \mode_leave_vertical:
2959       \arraybackslash
2960       \vrule height \box_ht:N \@arstrutbox depth 0 pt width 0 pt
2961     }
2962     c
2963     < {
2964       \vrule height 0 pt depth \box_dp:N \@arstrutbox width 0 pt
2965       \end { minipage }
2966       \@@_cell_end:
2967     }
2968   }

```

We test for the presence of a <.

```

2969   \@@_make_m_preamble_x:n
2970 }

```

For w and W

```

2971 \cs_new_protected:Npn \@@_make_m_preamble_v:nnnn #1 #2 #3 #4
2972 {
2973   \tl_gput_right:Nn \g_@@_preamble_tl
2974   {
2975     > {
2976       \dim_set:Nn \l_@@_col_width_dim { #4 }
2977       \hbox_set:Nw \l_@@_cell_box
2978       \@@_cell_begin:w
2979       \cs_set_nopar:Npn \l_@@_hpos_cell_tl { #3 }
2980     }
2981     c
2982     < {
2983       \@@_cell_end:
2984       \hbox_set_end:
2985       \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
2986       #1
2987       \@@_adjust_size_box:
2988       \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2989     }
2990   }

```

We test for the presence of a <.

```

2991   \@@_make_m_preamble_x:n
2992 }

```

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

```

2993 \cs_new_protected:Npn \@@_make_m_preamble_x:n #1
2994 {
2995   \str_if_eq:nnTF { #1 } { < }
2996   \@@_make_m_preamble_ix:n
2997   { \@@_make_m_preamble:n { #1 } }
2998 }
2999 \cs_new_protected:Npn \@@_make_m_preamble_ix:n #1
3000 {
3001   \tl_gput_right:Nn \g_@@_preamble_tl { < { #1 } }
3002   \@@_make_m_preamble_x:n
3003 }

```

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`).

```

3004 \cs_new_protected:Npn \@@_put_box_in_flow:
3005 {
3006   \box_set_ht:Nn \l_tmpa_box { \box_ht:N \l_tmpa_box + \l_tmpa_dim }
3007   \box_set_dp:Nn \l_tmpa_box { \box_dp:N \l_tmpa_box + \l_tmpb_dim }
3008   \tl_if_eq:NNTF \l_@@_baseline_tl \c_@@_c_tl
3009   { \box_use_drop:N \l_tmpa_box }
3010   \@@_put_box_in_flow_i:
3011 }

```

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).

```

3012 \cs_new_protected:Npn \@@_put_box_in_flow_i:
3013 {
3014   \pgfpicture
3015   \@@_qpoint:n { row - 1 }
3016   \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3017   \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
3018   \dim_gadd:Nn \g_tmpa_dim \pgf@y
3019   \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).

```

3020   \tl_if_in:NnTF \l_@@_baseline_tl { line- }
3021   {
3022     \int_set:Nn \l_tmpa_int
3023     {
3024       \str_range:Nnn
3025       \l_@@_baseline_tl
3026       6
3027       { \tl_count:o \l_@@_baseline_tl }
3028     }
3029     \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
3030   }
3031   {
3032     \tl_if_eq:NnTF \l_@@_baseline_tl { t }
3033     { \int_set_eq:NN \l_tmpa_int \c_one_int }
3034     {
3035       \tl_if_eq:NnTF \l_@@_baseline_tl { b }
3036       { \int_set_eq:NN \l_tmpa_int \c@iRow }
3037       { \int_set:Nn \l_tmpa_int \l_@@_baseline_tl }
3038     }
3039     \bool_lazy_or:nnT
3040     { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
3041     { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }

```

```

3042         {
3043             \@@_error:n { bad-value-for-baseline }
3044             \int_set_eq:NN \l_tmpa_int \c_one_int
3045         }
3046         \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }

```

We take into account the position of the mathematical axis.

```

3047         \dim_gsub:Nn \g_tmpa_dim { \fontdimen22 \textfont2 }
3048     }
3049     \dim_gsub:Nn \g_tmpa_dim \pgf@y

```

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

```

3050     \endpgfpicture
3051     \box_move_up:nn \g_tmpa_dim { \box_use_drop:N \l_tmpa_box }
3052     \box_use_drop:N \l_tmpa_box
3053 }

```

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).

```

3054 \cs_new_protected:Npn \@@_use_arraybox_with_notes_c:
3055 {

```

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.

```

3056     \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3057     {
3058         \int_compare:nNnT \c_jCol > \c_one_int
3059         {
3060             \box_set_wd:Nn \l_@@_the_array_box
3061             { \box_wd:N \l_@@_the_array_box - \arraycolsep }
3062         }
3063     }

```

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

```

3064     \begin { minipage } [ t ] { \box_wd:N \l_@@_the_array_box }
3065     \bool_if:NT \l_@@_caption_above_bool
3066     {
3067         \tl_if_empty:NF \l_@@_caption_tl
3068         {
3069             \bool_set_false:N \g_@@_caption_finished_bool
3070             \int_gzero:N \c@tabularnote
3071             \@@_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).

```

3072         \int_compare:nNnT \g_@@_notes_caption_int > \c_zero_int
3073         {
3074             \tl_gput_right:Nx \g_@@_aux_tl
3075             {
3076                 \tl_set:Nn \exp_not:N \l_@@_note_in_caption_tl
3077                 { \int_use:N \g_@@_notes_caption_int }
3078             }
3079             \int_gzero:N \g_@@_notes_caption_int
3080         }
3081     }
3082 }

```

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

```

3083   \hbox
3084   {
3085     \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.

```

3086     \@@_create_extra_nodes:
3087     \seq_if_empty:NF \g_@@_blocks_seq \@@_draw_blocks:
3088   }

```

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

```

3089   \bool_lazy_any:nT
3090   {
3091     { ! \seq_if_empty_p:N \g_@@_notes_seq }
3092     { ! \seq_if_empty_p:N \g_@@_notes_in_caption_seq }
3093     { ! \tl_if_empty_p:o \g_@@_tabularnote_tl }
3094   }
3095   \@@_insert_tabularnotes:
3096   \cs_set_eq:NN \tabularnote \@@_tabularnote_error:n
3097   \bool_if:NF \l_@@_caption_above_bool \@@_insert_caption:
3098   \end { minipage }
3099 }

```

```

3100 \cs_new_protected:Npn \@@_insert_caption:
3101 {
3102   \tl_if_empty:NF \l_@@_caption_tl
3103   {
3104     \cs_if_exist:NTF \capttype
3105     { \@@_insert_caption_i: }
3106     { \@@_error:n { caption~outside~float } }
3107   }
3108 }

```

```

3109 \cs_new_protected:Npn \@@_insert_caption_i:
3110 {
3111   \group_begin:

```

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

```

3112   \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.

```

3113   \IfPackageLoadedTF { floatrow }
3114   { \cs_set_eq:NN \@makecaption \FR@makecaption }
3115   { }
3116   \tl_if_empty:NTF \l_@@_short_caption_tl
3117   { \caption }
3118   { \caption [ \l_@@_short_caption_tl ] }
3119   { \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.

```

3120   \bool_if:NF \g_@@_caption_finished_bool
3121   {
3122     \bool_gset_true:N \g_@@_caption_finished_bool
3123     \int_gset_eq:NN \g_@@_notes_caption_int \c@tabulernote
3124     \int_gzero:N \c@tabulernote
3125   }
3126   \tl_if_empty:NF \l_@@_label_tl { \label { \l_@@_label_tl } }
3127   \group_end:
3128 }

3129 \cs_new_protected:Npn \@@_tabulernote_error:n #1
3130 {
3131   \@@_error_or_warning:n { tabulernote~below~the~tabular }
3132   \@@_gredirect_none:n { tabulernote~below~the~tabular }
3133 }

3134 \cs_new_protected:Npn \@@_insert_tabularnotes:
3135 {
3136   \seq_gconcat:NNN \g_@@_notes_seq \g_@@_notes_in_caption_seq \g_@@_notes_seq
3137   \int_set:Nn \c@tabulernote { \seq_count:N \g_@@_notes_seq }
3138   \skip_vertical:N 0.65ex

```

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

```

3139   \group_begin:
3140   \l_@@_notes_code_before_tl
3141   \tl_if_empty:NF \g_@@_tabulernote_tl
3142   {
3143     \g_@@_tabulernote_tl \par
3144     \tl_gclear:N \g_@@_tabulernote_tl
3145   }

```

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.

```

3146   \int_compare:nNnT \c@tabulernote > \c_zero_int
3147   {
3148     \bool_if:NTF \l_@@_notes_para_bool
3149     {
3150       \begin { tabularnotes* }
3151       \seq_map_inline:Nn \g_@@_notes_seq
3152       { \@@_one_tabulernote:nn ##1 }
3153       \strut
3154       \end { tabularnotes* }

```

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

```

3155       \par
3156     }
3157   {
3158     \tabularnotes
3159     \seq_map_inline:Nn \g_@@_notes_seq
3160     { \@@_one_tabulernote:nn ##1 }
3161     \strut
3162     \endtabularnotes
3163   }
3164 }
3165 \unskip
3166 \group_end:
3167 \bool_if:NT \l_@@_notes_bottomrule_bool
3168 {
3169   \IfPackageLoadedTF { booktabs }
3170   {

```

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

```

3171       \skip_vertical:N \aboverulesep

```


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

```

3172         { \CT@arc@ \hrule height \heavyrulewidth }
3173     }
3174     { \@@_error_or_warning:n { bottomrule-without-booktabs } }
3175 }
3176 \l_@@_notes_code_after_tl
3177 \seq_gclear:N \g_@@_notes_seq
3178 \seq_gclear:N \g_@@_notes_in_caption_seq
3179 \int_gzero:N \c@tabularnote
3180 }

```

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.

```

3181 \cs_set_protected:Npn \@@_one_tabularnote:nn #1
3182 {
3183     \tl_if_novalue:nTF { #1 }
3184     { \item }
3185     { \item [ \@@_notes_label_in_list:n { #1 } ] }
3186 }

```

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.

```

3187 \cs_new_protected:Npn \@@_use_arraybox_with_notes_b:
3188 {
3189     \pgfpicture
3190     \@@_qpoint:n { row - 1 }
3191     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3192     \@@_qpoint:n { row - \int_use:N \c@iRow - base }
3193     \dim_gsub:Nn \g_tmpa_dim \pgf@y
3194     \endpgfpicture
3195     \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3196     \int_if_zero:nT \l_@@_first_row_int
3197     {
3198         \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3199         \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3200     }
3201     \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3202 }

```

Now, the general case.

```

3203 \cs_new_protected:Npn \@@_use_arraybox_with_notes:
3204 {

```

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

```

3205     \tl_if_eq:NnT \l_@@_baseline_tl { t }
3206     { \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`.

```

3207     \pgfpicture
3208     \@@_qpoint:n { row - 1 }
3209     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3210     \str_if_in:NnTF \l_@@_baseline_tl { line- }
3211     {
3212         \int_set:Nn \l_tmpa_int
3213         {
3214             \str_range:Nnn
3215             \l_@@_baseline_tl
3216             6
3217             { \tl_count:o \l_@@_baseline_tl }

```

```

3218     }
3219     \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
3220 }
3221 {
3222     \int_set:Nn \l_tmpa_int \l_@@_baseline_tl
3223     \bool_lazy_or:nnT
3224     { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
3225     { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
3226     {
3227         \@@_error:n { bad-value-for-baseline }
3228         \int_set:Nn \l_tmpa_int 1
3229     }
3230     \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }
3231 }
3232 \dim_gsub:Nn \g_tmpa_dim \pgf@y
3233 \endpgfpicture
3234 \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3235 \int_if_zero:nT \l_@@_first_row_int
3236 {
3237     \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3238     \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3239 }
3240 \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3241 }

```

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.

```

3242 \cs_new_protected:Npn \@@_put_box_in_flow_bis:nn #1 #2
3243 {

```

We will compute the real width of both delimiters used.

```

3244     \dim_zero_new:N \l_@@_real_left_delim_dim
3245     \dim_zero_new:N \l_@@_real_right_delim_dim
3246     \hbox_set:Nn \l_tmpb_box
3247     {
3248         \c_math_toggle_token
3249         \left #1
3250         \vcenter
3251         {
3252             \vbox_to_ht:nn
3253             { \box_ht_plus_dp:N \l_tmpa_box }
3254             { }
3255         }
3256         \right .
3257         \c_math_toggle_token
3258     }
3259     \dim_set:Nn \l_@@_real_left_delim_dim
3260     { \box_wd:N \l_tmpb_box - \nulldelimiterspace }
3261     \hbox_set:Nn \l_tmpb_box
3262     {
3263         \c_math_toggle_token
3264         \left .
3265         \vbox_to_ht:nn
3266         { \box_ht_plus_dp:N \l_tmpa_box }
3267         { }
3268         \right #2
3269         \c_math_toggle_token
3270     }
3271     \dim_set:Nn \l_@@_real_right_delim_dim
3272     { \box_wd:N \l_tmpb_box - \nulldelimiterspace }

```

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

```

3273     \skip_horizontal:N \l_@@_left_delim_dim

```

```

3274 \skip_horizontal:N -\l_@@_real_left_delim_dim
3275 \@@_put_box_in_flow:
3276 \skip_horizontal:N \l_@@_right_delim_dim
3277 \skip_horizontal:N -\l_@@_real_right_delim_dim
3278 }

```

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).

```

3279 \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).

```

3280 {
3281   \peek_remove_spaces:n
3282   {
3283     \peek_meaning:NTF \end
3284     \@@_analyze_end:Nn
3285     {
3286       \@@_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`).

```

3287       \exp_args:No \@@_array: \g_@@_array_preamble_tl
3288     }
3289   }
3290 }
3291 {
3292   \@@_create_col_nodes:
3293   \endarray
3294 }

```

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

```

3295 \NewDocumentEnvironment { @@-light-syntax } { b }
3296 {

```

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`.

```

3297   \tl_if_empty:nT { #1 } { \@@_fatal:n { empty-environment } }
3298   \tl_map_inline:nn { #1 }
3299   {
3300     \str_if_eq:nnT { ##1 } { & }
3301     { \@@_fatal:n { ampersand-in~light-syntax } }
3302     \str_if_eq:nnT { ##1 } { \ }
3303     { \@@_fatal:n { double-backslash-in~light-syntax } }
3304   }

```

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`.

```

3305   \@@_light_syntax_i:w #1 \CodeAfter \q_stop

```

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

```

3306 }

```

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.

```

3307 {
3308   \@@_create_col_nodes:
3309   \endarray
3310 }
3311 \cs_new_protected:Npn \@@_light_syntax_i:w #1\CodeAfter #2\q_stop
3312 {
3313   \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).

```

3314   \seq_clear_new:N \l_@@_rows_seq

```

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

```

3315   \tl_set_rescan:Nno \l_@@_end_of_row_tl { } \l_@@_end_of_row_tl
3316   \bool_if:NTF \l_@@_light_syntax_expanded_bool
3317     \seq_set_split:Nee
3318     \seq_set_split:Non
3319     \l_@@_rows_seq \l_@@_end_of_row_tl { #1 }

```

We delete the last row if it is empty.

```

3320   \seq_pop_right:NN \l_@@_rows_seq \l_tmpa_tl
3321   \tl_if_empty:NF \l_tmpa_tl
3322   { \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.

```

3323   \int_compare:nNnT \l_@@_last_row_int = { -1 }
3324     { \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`).

```

3325   \tl_build_begin:N \l_@@_new_body_tl
3326   \int_zero_new:N \l_@@_nb_cols_int

```

First, we treat the first row.

```

3327   \seq_pop_left:NN \l_@@_rows_seq \l_tmpa_tl
3328   \@@_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).

```

3329   \seq_map_inline:Nn \l_@@_rows_seq
3330   {
3331     \tl_build_put_right:Nn \l_@@_new_body_tl { \\ }
3332     \@@_line_with_light_syntax:n { ##1 }
3333   }
3334   \tl_build_end:N \l_@@_new_body_tl
3335   \int_compare:nNnT \l_@@_last_col_int = { -1 }
3336   {
3337     \int_set:Nn \l_@@_last_col_int
3338       { \l_@@_nb_cols_int - 1 + \l_@@_first_col_int }
3339   }

```

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.

```

3340   \@@_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`).

```

3341   \exp_args:No \@@_array: \g_@@_array_preamble_tl \l_@@_new_body_tl
3342 }

```

```

3343 \cs_new_protected:Npn \@@_line_with_light_syntax:n #1
3344 {
3345   \seq_clear_new:N \l_@@_cells_seq
3346   \seq_set_split:Nnn \l_@@_cells_seq { ~ } { #1 }
3347   \int_set:Nn \l_@@_nb_cols_int
3348   {
3349     \int_max:nn
3350     \l_@@_nb_cols_int
3351     { \seq_count:N \l_@@_cells_seq }
3352   }
3353   \seq_pop_left:NN \l_@@_cells_seq \l_tmpa_tl
3354   \exp_args:NNo \tl_build_put_right:Nn \l_@@_new_body_tl \l_tmpa_tl
3355   \seq_map_inline:Nn \l_@@_cells_seq
3356   { \tl_build_put_right:Nn \l_@@_new_body_tl { & ##1 } }
3357 }
3358 \cs_generate_variant:Nn \@@_line_with_light_syntax:n { o }

```

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.

```

3359 \cs_new_protected:Npn \@@_analyze_end:Nn #1 #2
3360 {
3361   \str_if_eq:onT \g_@@_name_env_str { #2 }
3362   { \@@_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.

```

3363   \end { #2 }
3364 }

```

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).

```

3365 \cs_new:Npn \@@_create_col_nodes:
3366 {
3367   \crrr
3368   \int_if_zero:nT \l_@@_first_col_int
3369   {
3370     \omit
3371     \hbox_overlap_left:n
3372     {
3373       \bool_if:NT \l_@@_code_before_bool
3374       { \pgfsys@markposition { \@@_env: - col - 0 } }
3375       \pgfpicture
3376       \pgfrememberpicturepositiononpagetrue
3377       \pgfcoordinate { \@@_env: - col - 0 } \pgfpointorigin
3378       \str_if_empty:NF \l_@@_name_str
3379       { \pgfnodealias { \l_@@_name_str - col - 0 } { \@@_env: - col - 0 } }
3380       \endpgfpicture
3381       \skip_horizontal:N 2\col@sep
3382       \skip_horizontal:N \g_@@_width_first_col_dim
3383     }
3384     &
3385   }
3386   \omit

```

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

```

3387   \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).

```

3388   \int_if_zero:nTF \l_@@_first_col_int
3389   {

```

```

3390 \bool_if:NT \l_@@_code_before_bool
3391 {
3392   \hbox
3393   {
3394     \skip_horizontal:N -0.5\arrayrulewidth
3395     \pgfsys@markposition { \@@_env: - col - 1 }
3396     \skip_horizontal:N 0.5\arrayrulewidth
3397   }
3398 }
3399 \pgfpicture
3400 \pgfrememberpicturepositiononpagetrue
3401 \pgfcoordinate { \@@_env: - col - 1 }
3402 { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3403 \str_if_empty:NF \l_@@_name_str
3404 { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
3405 \endpgfpicture
3406 }
3407 {
3408 \bool_if:NT \l_@@_code_before_bool
3409 {
3410   \hbox
3411   {
3412     \skip_horizontal:N 0.5\arrayrulewidth
3413     \pgfsys@markposition { \@@_env: - col - 1 }
3414     \skip_horizontal:N -0.5\arrayrulewidth
3415   }
3416 }
3417 \pgfpicture
3418 \pgfrememberpicturepositiononpagetrue
3419 \pgfcoordinate { \@@_env: - col - 1 }
3420 { \pgfpoint { 0.5 \arrayrulewidth } \c_zero_dim }
3421 \str_if_empty:NF \l_@@_name_str
3422 { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
3423 \endpgfpicture
3424 }

```

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.

```

3425 \skip_gset:Nn \g_tmpa_skip { 0 pt~plus 1 fill }
3426 \bool_if:NF \l_@@_auto_columns_width_bool
3427 { \dim_compare:nNnT \l_@@_columns_width_dim > \c_zero_dim }
3428 {
3429   \bool_lazy_and:nnTF
3430   \l_@@_auto_columns_width_bool
3431   { \bool_not_p:n \l_@@_block_auto_columns_width_bool }
3432   { \skip_gadd:Nn \g_tmpa_skip \g_@@_max_cell_width_dim }
3433   { \skip_gadd:Nn \g_tmpa_skip \l_@@_columns_width_dim }
3434   \skip_gadd:Nn \g_tmpa_skip { 2 \col@sep }
3435 }
3436 \skip_horizontal:N \g_tmpa_skip
3437 \hbox
3438 {
3439   \bool_if:NT \l_@@_code_before_bool
3440   {
3441     \hbox
3442     {
3443       \skip_horizontal:N -0.5\arrayrulewidth
3444       \pgfsys@markposition { \@@_env: - col - 2 }
3445       \skip_horizontal:N 0.5\arrayrulewidth

```

```

3446     }
3447   }
3448   \pgfpicture
3449   \pgfrememberpicturepositiononpagetrue
3450   \pgfcoordinate { \@@_env: - col - 2 }
3451   { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3452   \str_if_empty:NF \l_@@_name_str
3453   { \pgfnodealias { \l_@@_name_str - col - 2 } { \@@_env: - col - 2 } }
3454   \endpgfpicture
3455 }

```

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.

```

3456   \int_gset_eq:NN \g_tmpa_int \c_one_int
3457   \bool_if:NTF \g_@@_last_col_found_bool
3458   { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 3 } \c_zero_int } }
3459   { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 2 } \c_zero_int } }
3460   {
3461     &
3462     \omit
3463     \int_gincr:N \g_tmpa_int

```

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

```

3464     \skip_horizontal:N \g_tmpa_skip
3465     \bool_if:NT \l_@@_code_before_bool
3466     {
3467       \hbox
3468       {
3469         \skip_horizontal:N -0.5\arrayrulewidth
3470         \pgfsys@markposition
3471         { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3472         \skip_horizontal:N 0.5\arrayrulewidth
3473       }
3474     }

```

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

```

3475   \pgfpicture
3476   \pgfrememberpicturepositiononpagetrue
3477   \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3478   { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3479   \str_if_empty:NF \l_@@_name_str
3480   {
3481     \pgfnodealias
3482     { \l_@@_name_str - col - \int_eval:n { \g_tmpa_int + 1 } }
3483     { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3484   }
3485   \endpgfpicture
3486 }

3487 &
3488 \omit

```

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

```

3489   \int_if_zero:nT \g_@@_col_total_int
3490   { \skip_gset:Nn \g_tmpa_skip { 0 pt-plus 1 fill } }
3491   \skip_horizontal:N \g_tmpa_skip
3492   \int_gincr:N \g_tmpa_int
3493   \bool_lazy_any:nF
3494   {
3495     \g_@@_delims_bool
3496     \l_@@_tabular_bool
3497     { ! \clist_if_empty_p:N \l_@@_vlines_clist }
3498     \l_@@_exterior_arraycolsep_bool

```

```

3499         \l_@@_bar_at_end_of_pream_bool
3500     }
3501     { \skip_horizontal:N -\col@sep }
3502 \bool_if:NT \l_@@_code_before_bool
3503     {
3504         \hbox
3505         {
3506             \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.

```

3507         \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3508         { \skip_horizontal:N -\arraycolsep }
3509         \pgfsys@markposition
3510         { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3511         \skip_horizontal:N 0.5\arrayrulewidth
3512         \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3513         { \skip_horizontal:N \arraycolsep }
3514     }
3515 }
3516 \pgfpicture
3517 \pgfrememberpicturerepositiononpagetrue
3518 \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3519 {
3520     \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3521     {
3522         \pgfpoint
3523         { - 0.5 \arrayrulewidth - \arraycolsep }
3524         \c_zero_dim
3525     }
3526     { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3527 }
3528 \str_if_empty:NF \l_@@_name_str
3529 {
3530     \pgfnodealias
3531     { \l_@@_name_str - col - \int_eval:n { \g_tmpa_int + 1 } }
3532     { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3533 }
3534 \endpgfpicture

3535 \bool_if:NT \g_@@_last_col_found_bool
3536 {
3537     \hbox_overlap_right:n
3538     {
3539         \skip_horizontal:N \g_@@_width_last_col_dim
3540         \skip_horizontal:N \col@sep
3541         \bool_if:NT \l_@@_code_before_bool
3542         {
3543             \pgfsys@markposition
3544             { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3545         }
3546         \pgfpicture
3547         \pgfrememberpicturerepositiononpagetrue
3548         \pgfcoordinate
3549         { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3550         \pgfpointorigin
3551         \str_if_empty:NF \l_@@_name_str
3552         {
3553             \pgfnodealias
3554             {
3555                 \l_@@_name_str - col
3556                 - \int_eval:n { \g_@@_col_total_int + 1 }

```



```

3557         }
3558         { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3559     }
3560     \endpgfpicture
3561 }
3562 }
3563 % \cr
3564 }

```

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

```

3565 \tl_const:Nn \c_@@_preamble_first_col_tl
3566 {
3567     >
3568     {

```

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).

```

3569         \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
3570         \bool_gset_true:N \g_@@_after_col_zero_bool
3571         \@@_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.

```

3572         \hbox_set:Nw \l_@@_cell_box
3573         \@@_math_toggle:
3574         \@@_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”.

```

3575         \int_compare:nNnT \c@iRow > \c_zero_int
3576         {
3577             \bool_lazy_or:nnT
3578             { \int_compare_p:nNn \l_@@_last_row_int < \c_zero_int }
3579             { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
3580             {
3581                 \l_@@_code_for_first_col_tl
3582                 \xglobal \colorlet { nicematrix-first-col } { . }
3583             }
3584         }
3585     }

```

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`.

```

3586         l
3587         <
3588         {
3589             \@@_math_toggle:
3590             \hbox_set_end:
3591             \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
3592             \@@_adjust_size_box:
3593             \@@_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.

```

3594         \dim_gset:Nn \g_@@_width_first_col_dim
3595         { \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.

```

3596         \hbox_overlap_left:n
3597         {
3598             \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > \c_zero_dim
3599             \@@_node_for_cell:
3600             { \box_use_drop:N \l_@@_cell_box }
3601             \skip_horizontal:N \l_@@_left_delim_dim

```

```

3602         \skip_horizontal:N \l_@@_left_margin_dim
3603         \skip_horizontal:N \l_@@_extra_left_margin_dim
3604     }
3605     \bool_gset_false:N \g_@@_empty_cell_bool
3606     \skip_horizontal:N -2\col@sep
3607 }
3608 }

```

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

```

3609 \tl_const:Nn \c_@@_preamble_last_col_tl
3610 {
3611     >
3612     {
3613         \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).

```

3614         \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.

```

3615         \bool_gset_true:N \g_@@_last_col_found_bool
3616         \int_gincr:N \c@jCol
3617         \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.

```

3618         \hbox_set:Nw \l_@@_cell_box
3619         \@@_math_toggle:
3620         \@@_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”.

```

3621         \int_compare:nNnT \c@iRow > \c_zero_int
3622         {
3623             \bool_lazy_or:nnT
3624             { \int_compare_p:nNn \l_@@_last_row_int < \c_zero_int }
3625             { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
3626             {
3627                 \l_@@_code_for_last_col_tl
3628                 \xglobal \colorlet { nicematrix-last-col } { . }
3629             }
3630         }
3631     }
3632     1
3633     <
3634     {
3635         \@@_math_toggle:
3636         \hbox_set_end:
3637         \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
3638         \@@_adjust_size_box:
3639         \@@_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.

```

3640         \dim_gset:Nn \g_@@_width_last_col_dim
3641         { \dim_max:nn \g_@@_width_last_col_dim { \box_wd:N \l_@@_cell_box } }
3642         \skip_horizontal:N -2\col@sep

```

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

```

3643         \hbox_overlap_right:n
3644         {
3645             \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \c_zero_dim
3646             {
3647                 \skip_horizontal:N \l_@@_right_delim_dim
3648                 \skip_horizontal:N \l_@@_right_margin_dim
3649                 \skip_horizontal:N \l_@@_extra_right_margin_dim

```

```

3650         \@@_node_for_cell:
3651     }
3652 }
3653 \bool_gset_false:N \g_@@_empty_cell_bool
3654 }
3655 }

```

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

```

3656 \NewDocumentEnvironment { NiceArray } { }
3657 {
3658     \bool_gset_false:N \g_@@_delims_bool
3659     \str_if_empty:NT \g_@@_name_env_str
3660     { \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).

```

3661     \NiceArrayWithDelims . .
3662 }
3663 { \endNiceArrayWithDelims }

```

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

```

3664 \cs_new_protected:Npn \@@_def_env:nnn #1 #2 #3
3665 {
3666     \NewDocumentEnvironment { #1 NiceArray } { }
3667     {
3668         \bool_gset_true:N \g_@@_delims_bool
3669         \str_if_empty:NT \g_@@_name_env_str
3670         { \str_gset:Nn \g_@@_name_env_str { #1 NiceArray } }
3671         \@@_test_if_math_mode:
3672         \NiceArrayWithDelims #2 #3
3673     }
3674     { \endNiceArrayWithDelims }
3675 }
3676 \@@_def_env:nnn p ( )
3677 \@@_def_env:nnn b [ ]
3678 \@@_def_env:nnn B \{ \}
3679 \@@_def_env:nnn v | |
3680 \@@_def_env:nnn V \| \|

```

14 The environment `{NiceMatrix}` and its variants

```

3681 \cs_new_protected:Npn \@@_begin_of_NiceMatrix:nn #1 #2
3682 {
3683     \bool_set_false:N \l_@@_preamble_bool
3684     \tl_clear:N \l_tmpa_tl
3685     \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3686     { \tl_set:Nn \l_tmpa_tl { @ { } } }
3687     \tl_put_right:Nn \l_tmpa_tl
3688     {
3689         *
3690         {
3691             \int_case:nnF \l_@@_last_col_int
3692             {
3693                 { -2 } { \c@MaxMatrixCols }
3694                 { -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).

```

3695         }
3696         { \int_eval:n { \l_@@_last_col_int - 1 } }
3697     }
3698     { #2 }
3699 }
3700 \tl_set:Nn \l_tmpb_tl { \use:c { #1 NiceArray } }
3701 \exp_args:No \l_tmpb_tl \l_tmpa_tl
3702 }
3703 \cs_generate_variant:Nn \@@_begin_of_NiceMatrix:nn { n V }
3704 \clist_map_inline:nn { p , b , B , v , V }
3705 {
3706     \NewDocumentEnvironment { #1 NiceMatrix } { ! 0 { } }
3707     {
3708         \bool_gset_true:N \g_@@_delims_bool
3709         \str_gset:Nn \g_@@_name_env_str { #1 NiceMatrix }
3710         \int_if_zero:nT \l_@@_last_col_int
3711         {
3712             \bool_set_true:N \l_@@_last_col_without_value_bool
3713             \int_set:Nn \l_@@_last_col_int { -1 }
3714         }
3715         \keys_set:nn { NiceMatrix / NiceMatrix } { ##1 }
3716         \@@_begin_of_NiceMatrix:nV { #1 } \l_@@_columns_type_tl
3717     }
3718     { \use:c { end #1 NiceArray } }
3719 }

```

We define also an environment {NiceMatrix}

```

3720 \NewDocumentEnvironment { NiceMatrix } { ! 0 { } }
3721 {
3722     \str_gset:Nn \g_@@_name_env_str { NiceMatrix }
3723     \int_if_zero:nT \l_@@_last_col_int
3724     {
3725         \bool_set_true:N \l_@@_last_col_without_value_bool
3726         \int_set:Nn \l_@@_last_col_int { -1 }
3727     }
3728     \keys_set:nn { NiceMatrix / NiceMatrix } { #1 }
3729     \bool_lazy_or:nnT
3730     { \clist_if_empty_p:N \l_@@_vlines_clist }
3731     { \l_@@_except_borders_bool }
3732     { \bool_set_true:N \l_@@_NiceMatrix_without_vlines_bool }
3733     \@@_begin_of_NiceMatrix:nV { } \l_@@_columns_type_tl
3734 }
3735 { \endNiceArray }

```

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

```

3736 \cs_new_protected:Npn \@@_NotEmpty:
3737 { \bool_gset_true:N \g_@@_not_empty_cell_bool }

```

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

```

3738 \NewDocumentEnvironment { NiceTabular } { 0 { } m ! 0 { } }
3739 {

```

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.

```

3740     \dim_compare:nNnT \l_@@_width_dim = \c_zero_dim
3741     { \dim_set_eq:NN \l_@@_width_dim \linewidth }
3742     \str_gset:Nn \g_@@_name_env_str { NiceTabular }
3743     \keys_set:nn { NiceMatrix / NiceTabular } { #1 , #3 }
3744     \tl_if_empty:NF \l_@@_short_caption_tl
3745     {

```

```

3746     \tl_if_empty:NT \l_@@_caption_tl
3747     {
3748         \@@_error_or_warning:n { short-caption-without~caption }
3749         \tl_set_eq:NN \l_@@_caption_tl \l_@@_short_caption_tl
3750     }
3751 }
3752 \tl_if_empty:NF \l_@@_label_tl
3753 {
3754     \tl_if_empty:NT \l_@@_caption_tl
3755     { \@@_error_or_warning:n { label~without~caption } }
3756 }
3757 \NewDocumentEnvironment { TabularNote } { b }
3758 {
3759     \bool_if:NTF \l_@@_in_code_after_bool
3760     { \@@_error_or_warning:n { TabularNote~in~CodeAfter } }
3761     {
3762         \tl_if_empty:NF \g_@@_tabularnote_tl
3763         { \tl_gput_right:Nn \g_@@_tabularnote_tl { \par } }
3764         \tl_gput_right:Nn \g_@@_tabularnote_tl { ##1 }
3765     }
3766 }
3767 { }
3768 \@@_settings_for_tabular:
3769 \NiceArray { #2 }
3770 }
3771 {
3772     \endNiceArray
3773     \bool_if:NT \c_@@_testphase_table_bool
3774     { \UseTaggingSocket { tbl / hmode / end } }
3775 }
3776 \cs_new_protected:Npn \@@_settings_for_tabular:
3777 {
3778     \bool_set_true:N \l_@@_tabular_bool
3779     \cs_set_eq:NN \@@_math_toggle: \prg_do_nothing:
3780     \cs_set_eq:NN \@@_tuning_not_tabular_begin: \prg_do_nothing:
3781     \cs_set_eq:NN \@@_tuning_not_tabular_end: \prg_do_nothing:
3782 }

3783 \NewDocumentEnvironment { NiceTabularX } { m O { } m ! O { } }
3784 {
3785     \str_gset:Nn \g_@@_name_env_str { NiceTabularX }
3786     \dim_zero_new:N \l_@@_width_dim
3787     \dim_set:Nn \l_@@_width_dim { #1 }
3788     \keys_set:nn { NiceMatrix / NiceTabular } { #2 , #4 }
3789     \@@_settings_for_tabular:
3790     \NiceArray { #3 }
3791 }
3792 {
3793     \endNiceArray
3794     \int_if_zero:nT \g_@@_total_X_weight_int
3795     { \@@_error:n { NiceTabularX~without~X } }
3796 }

3797 \NewDocumentEnvironment { NiceTabular* } { m O { } m ! O { } }
3798 {
3799     \str_gset:Nn \g_@@_name_env_str { NiceTabular* }
3800     \dim_set:Nn \l_@@_tabular_width_dim { #1 }
3801     \keys_set:nn { NiceMatrix / NiceTabular } { #2 , #4 }
3802     \@@_settings_for_tabular:
3803     \NiceArray { #3 }
3804 }
3805 { \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`).

```

3806 \cs_new_protected:Npn \@@_deal_with_rounded_corners:
3807 {
3808   \bool_lazy_all:nT
3809   {
3810     { \int_compare_p:nNn \l_@@_tab_rounded_corners_dim > \c_zero_dim }
3811     \l_@@_hvlines_bool
3812     { ! \g_@@_delims_bool }
3813     { ! \l_@@_except_borders_bool }
3814   }
3815   {
3816     \bool_set_true:N \l_@@_except_borders_bool
3817     \clist_if_empty:NF \l_@@_corners_clist
3818     { \@@_error:n { hvlines,~rounded-corners-and-corners } }
3819     \tl_gput_right:Nn \g_@@_pre_code_after_tl
3820     {
3821       \@@_stroke_block:nnn
3822       {
3823         rounded-corners = \dim_use:N \l_@@_tab_rounded_corners_dim ,
3824         draw = \l_@@_rules_color_tl
3825       }
3826       { 1-1 }
3827       { \int_use:N \c@iRow - \int_use:N \c@jCol }
3828     }
3829   }
3830 }

3831 \cs_new_protected:Npn \@@_after_array:
3832 {

```

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`.

```

3833   \hook_gremove_code:nn { env / tabular / begin } { nicematrix }
3834   \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.

```

3835   \bool_if:NT \g_@@_last_col_found_bool
3836   { \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`.

```

3837   \bool_if:NT \l_@@_last_col_without_value_bool
3838   { \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.

```

3839   \bool_if:NT \l_@@_last_row_without_value_bool
3840   { \int_set_eq:NN \l_@@_last_row_int \g_@@_row_total_int }

```

```

3841 \tl_gput_right:Nx \g_@@_aux_tl
3842 {
3843   \seq_gset_from_clist:Nn \exp_not:N \g_@@_size_seq
3844   {
3845     \int_use:N \l_@@_first_row_int ,
3846     \int_use:N \c@iRow ,
3847     \int_use:N \g_@@_row_total_int ,
3848     \int_use:N \l_@@_first_col_int ,
3849     \int_use:N \c@jCol ,
3850     \int_use:N \g_@@_col_total_int
3851   }
3852 }

```

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`).

```

3853 \seq_if_empty:NF \g_@@_pos_of_blocks_seq
3854 {
3855   \tl_gput_right:Nx \g_@@_aux_tl
3856   {
3857     \seq_gset_from_clist:Nn \exp_not:N \g_@@_pos_of_blocks_seq
3858     { \seq_use:Nnnn \g_@@_pos_of_blocks_seq , , , }
3859   }
3860 }
3861 \seq_if_empty:NF \g_@@_multicolumn_cells_seq
3862 {
3863   \tl_gput_right:Nx \g_@@_aux_tl
3864   {
3865     \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_cells_seq
3866     { \seq_use:Nnnn \g_@@_multicolumn_cells_seq , , , }
3867     \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_sizes_seq
3868     { \seq_use:Nnnn \g_@@_multicolumn_sizes_seq , , , }
3869   }
3870 }

```

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

```

3871 \@@_create_diag_nodes:

```

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

```

3872 \pgfpicture
3873 \int_step_inline:nn \c@iRow
3874 {
3875   \pgfnodealias
3876   { \@@_env: - ##1 - last }
3877   { \@@_env: - ##1 - \int_use:N \c@jCol }
3878 }
3879 \int_step_inline:nn \c@jCol
3880 {
3881   \pgfnodealias
3882   { \@@_env: - last - ##1 }
3883   { \@@_env: - \int_use:N \c@iRow - ##1 }
3884 }
3885 \str_if_empty:NF \l_@@_name_str
3886 {
3887   \int_step_inline:nn \c@iRow
3888   {
3889     \pgfnodealias
3890     { \l_@@_name_str - ##1 - last }
3891     { \@@_env: - ##1 - \int_use:N \c@jCol }
3892   }
3893   \int_step_inline:nn \c@jCol
3894   {
3895     \pgfnodealias

```

```

3896         { \l_@@_name_str - last - ##1 }
3897         { \@@_env: - \int_use:N \c@iRow - ##1 }
3898     }
3899 }
3900 \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.

```

3901 \bool_if:NT \l_@@_parallelize_diags_bool
3902 {
3903     \int_gzero_new:N \g_@@_ddots_int
3904     \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.

```

3905     \dim_gzero_new:N \g_@@_delta_x_one_dim
3906     \dim_gzero_new:N \g_@@_delta_y_one_dim
3907     \dim_gzero_new:N \g_@@_delta_x_two_dim
3908     \dim_gzero_new:N \g_@@_delta_y_two_dim
3909 }

3910 \int_zero_new:N \l_@@_initial_i_int
3911 \int_zero_new:N \l_@@_initial_j_int
3912 \int_zero_new:N \l_@@_final_i_int
3913 \int_zero_new:N \l_@@_final_j_int
3914 \bool_set_false:N \l_@@_initial_open_bool
3915 \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.

```

3916 \bool_if:NT \l_@@_small_bool
3917 {
3918     \dim_set:Nn \l_@@_xdots_radius_dim { 0.7 \l_@@_xdots_radius_dim }
3919     \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.

```

3920     \dim_set:Nn \l_@@_xdots_shorten_start_dim
3921     { 0.6 \l_@@_xdots_shorten_start_dim }
3922     \dim_set:Nn \l_@@_xdots_shorten_end_dim
3923     { 0.6 \l_@@_xdots_shorten_end_dim }
3924 }

```

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

```

3925 \@@_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.

```

3926 \@@_compute_corners:

```

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

```

3927 \@@_adjust_pos_of_blocks_seq:

```

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


```

3928 \@@_deal_with_rounded_corners:
3929 \tl_if_empty:NF \l_@@_hlines_clist \@@_draw_hlines:
3930 \tl_if_empty:NF \l_@@_vlines_clist \@@_draw_vlines:

```

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

```

3931 \IfPackageLoadedTF { tikz }
3932 {
3933   \tikzset
3934   {
3935     every~picture / .style =
3936     {
3937       overlay ,
3938       remember~picture ,
3939       name~prefix = \@@_env: -
3940     }
3941   }
3942 }
3943 { }
3944 \bool_if:NT \c_@@_tagging_array_bool
3945 { \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign: }
3946 \cs_set_eq:NN \SubMatrix \@@_SubMatrix
3947 \cs_set_eq:NN \UnderBrace \@@_UnderBrace
3948 \cs_set_eq:NN \OverBrace \@@_OverBrace
3949 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
3950 \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
3951 \cs_set_eq:NN \line \@@_line
3952 \g_@@_pre_code_after_tl
3953 \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.

```

3954 \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).

```

3955 \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`).

```

3956 % \int_compare:nNtT { \char_value_catcode:n { 60 } } = { 13 }
3957 % { \@@_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:`.

```

3958 \bool_set_true:N \l_@@_in_code_after_bool
3959 \exp_last_unbraced:No \@@_CodeAfter_keys: \g_nicematrix_code_after_tl
3960 \scan_stop:
3961 \tl_gclear:N \g_nicematrix_code_after_tl
3962 \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.

```

3963 \seq_if_empty:NF \g_@@_rowlistcolors_seq { \@@_clear_rowlistcolors_seq: }
3964 \tl_if_empty:NF \g_@@_pre_code_before_tl
3965 {
3966   \tl_gput_right:Nx \g_@@_aux_tl
3967   {
3968     \tl_gset:Nn \exp_not:N \g_@@_pre_code_before_tl

```

```

3969         { \exp_not:o \g_@@_pre_code_before_tl }
3970     }
3971     \tl_gclear:N \g_@@_pre_code_before_tl
3972 }
3973 \tl_if_empty:NF \g_nicematrix_code_before_tl
3974 {
3975     \tl_gput_right:Nx \g_@@_aux_tl
3976     {
3977         \tl_gset:Nn \exp_not:N \g_@@_code_before_tl
3978         { \exp_not:o \g_nicematrix_code_before_tl }
3979     }
3980     \tl_gclear:N \g_nicematrix_code_before_tl
3981 }

3982 \str_gclear:N \g_@@_name_env_str
3983 \@@_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`.

```

3984     \cs_gset_eq:NN \CT@arc@ \@@_old_CT@arc@
3985 }

```

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`.

```

3986 \NewDocumentCommand \@@_CodeAfter_keys: { 0 { } }
3987 { \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).

```

3988 \cs_new_protected:Npn \@@_adjust_pos_of_blocks_seq:
3989 {
3990     \seq_gset_map_x:NNn \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq
3991     { \@@_adjust_pos_of_blocks_seq_i:nnnnn ##1 }
3992 }

```

The following command must *not* be protected.

```

3993 \cs_new:Npn \@@_adjust_pos_of_blocks_seq_i:nnnnn #1 #2 #3 #4 #5
3994 {
3995     { #1 }
3996     { #2 }
3997     {
3998         \int_compare:nNnTF { #3 } > { 99 }
3999         { \int_use:N \c@iRow }
4000         { #3 }
4001     }
4002     {
4003         \int_compare:nNnTF { #4 } > { 99 }
4004         { \int_use:N \c@jCol }
4005         { #4 }
4006     }
4007     { #5 }
4008 }

```

¹²e.g. `\color[rgb]{0.5,0.5,0}`

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).

```

4009 \hook_gput_code:nnn { begindocument } { . }
4010 {
4011   \cs_new_protected:Npx \@@_draw_dotted_lines:
4012   {
4013     \c_@@_pgfortikzpicture_tl
4014     \@@_draw_dotted_lines_i:
4015     \c_@@_endpgfortikzpicture_tl
4016   }
4017 }

```

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

```

4018 \cs_new_protected:Npn \@@_draw_dotted_lines_i:
4019 {
4020   \pgfrememberpicturepositiononpagetrue
4021   \pgf@relevantforpicturesizefalse
4022   \g_@@_HVdotsfor_lines_tl
4023   \g_@@_Vdots_lines_tl
4024   \g_@@_Ddots_lines_tl
4025   \g_@@_Idots_lines_tl
4026   \g_@@_Cdots_lines_tl
4027   \g_@@_Ldots_lines_tl
4028 }

4029 \cs_new_protected:Npn \@@_restore_iRow_jCol:
4030 {
4031   \cs_if_exist:NT \theiRow { \int_gset_eq:NN \c@iRow \l_@@_old_iRow_int }
4032   \cs_if_exist:NT \thejCol { \int_gset_eq:NN \c@jCol \l_@@_old_jCol_int }
4033 }

```

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

```

4034 \pgfdeclareshape { @@_diag_node }
4035 {
4036   \savedanchor { \five }
4037   {
4038     \dim_gset_eq:NN \pgf@x \l_tmpa_dim
4039     \dim_gset_eq:NN \pgf@y \l_tmpb_dim
4040   }
4041   \anchor { 5 } { \five }
4042   \anchor { center } { \pgfpointorigin }
4043   \anchor { 1 } { \five \pgf@x = 0.2 \pgf@x \pgf@y = 0.2 \pgf@y }
4044   \anchor { 2 } { \five \pgf@x = 0.4 \pgf@x \pgf@y = 0.4 \pgf@y }
4045   \anchor { 3 } { \five \pgf@x = 0.6 \pgf@x \pgf@y = 0.6 \pgf@y }
4046   \anchor { 4 } { \five \pgf@x = 0.8 \pgf@x \pgf@y = 0.8 \pgf@y }
4047   \anchor { 6 } { \five \pgf@x = 1.2 \pgf@x \pgf@y = 1.2 \pgf@y }
4048   \anchor { 7 } { \five \pgf@x = 1.4 \pgf@x \pgf@y = 1.4 \pgf@y }
4049   \anchor { 8 } { \five \pgf@x = 1.6 \pgf@x \pgf@y = 1.6 \pgf@y }
4050   \anchor { 9 } { \five \pgf@x = 1.8 \pgf@x \pgf@y = 1.8 \pgf@y }
4051 }

```

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).

```

4052 \cs_new_protected:Npn \@@_create_diag_nodes:
4053 {
4054   \pgfpicture
4055   \pgfrememberpicturepositiononpagetrue
4056   \int_step_inline:nn { \int_max:nn \c@iRow \c@jCol }

```

```

4057 {
4058   \@@_qpoint:n { col - \int_min:nn { ##1 } { \c@jCol + 1 } }
4059   \dim_set_eq:NN \l_tmpa_dim \pgf@x
4060   \@@_qpoint:n { row - \int_min:nn { ##1 } { \c@iRow + 1 } }
4061   \dim_set_eq:NN \l_tmpb_dim \pgf@y
4062   \@@_qpoint:n { col - \int_min:nn { ##1 + 1 } { \c@jCol + 1 } }
4063   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
4064   \@@_qpoint:n { row - \int_min:nn { ##1 + 1 } { \c@iRow + 1 } }
4065   \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
4066   \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.

```

4067   \dim_set:Nn \l_tmpa_dim { ( \l_@@_tmpc_dim - \l_tmpa_dim ) / 2 }
4068   \dim_set:Nn \l_tmpb_dim { ( \l_@@_tmpd_dim - \l_tmpb_dim ) / 2 }
4069   \pgfnode { @@_diag_node } { center } { } { \@@_env: - ##1 } { }
4070   \str_if_empty:NF \l_@@_name_str
4071   { \pgfnodealias { \l_@@_name_str - ##1 } { \@@_env: - ##1 } }
4072 }

```

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

```

4073   \int_set:Nn \l_tmpa_int { \int_max:nn \c@iRow \c@jCol + 1 }
4074   \@@_qpoint:n { row - \int_min:nn { \l_tmpa_int } { \c@iRow + 1 } }
4075   \dim_set_eq:NN \l_tmpa_dim \pgf@y
4076   \@@_qpoint:n { col - \int_min:nn { \l_tmpa_int } { \c@jCol + 1 } }
4077   \pgfcoordinate
4078   { \@@_env: - \int_use:N \l_tmpa_int } { \pgfpoint \pgf@x \l_tmpa_dim }
4079   \pgfnodealias
4080   { \@@_env: - last }
4081   { \@@_env: - \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
4082   \str_if_empty:NF \l_@@_name_str
4083   {
4084     \pgfnodealias
4085     { \l_@@_name_str - \int_use:N \l_tmpa_int }
4086     { \@@_env: - \int_use:N \l_tmpa_int }
4087     \pgfnodealias
4088     { \l_@@_name_str - last }
4089     { \@@_env: - last }
4090   }
4091   \endpgfpicture
4092 }

```

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.

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

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

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

- the fourth argument is the y -value of the orientation vector of the line.

This command computes:

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

```
4093 \cs_new_protected:Npn \@@_find_extremities_of_line:nnnn #1 #2 #3 #4
4094 {
```

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

```
4095 \cs_set:cpn { @@ _ dotted _ #1 - #2 } { }
```

Initialization of variables.

```
4096 \int_set:Nn \l_@@_initial_i_int { #1 }
4097 \int_set:Nn \l_@@_initial_j_int { #2 }
4098 \int_set:Nn \l_@@_final_i_int { #1 }
4099 \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.

```
4100 \bool_set_false:N \l_@@_stop_loop_bool
4101 \bool_do_until:Nn \l_@@_stop_loop_bool
4102 {
4103   \int_add:Nn \l_@@_final_i_int { #3 }
4104   \int_add:Nn \l_@@_final_j_int { #4 }
```

We test if we are still in the matrix.

```
4105   \bool_set_false:N \l_@@_final_open_bool
4106   \int_compare:nNnTF \l_@@_final_i_int > \l_@@_row_max_int
4107   {
4108     \int_compare:nNnTF { #3 } = \c_one_int
4109     { \bool_set_true:N \l_@@_final_open_bool }
4110     {
4111       \int_compare:nNnT \l_@@_final_j_int > \l_@@_col_max_int
4112       { \bool_set_true:N \l_@@_final_open_bool }
4113     }
4114   }
4115   {
4116     \int_compare:nNnTF \l_@@_final_j_int < \l_@@_col_min_int
4117     {
4118       \int_compare:nNnT { #4 } = { -1 }
4119       { \bool_set_true:N \l_@@_final_open_bool }
4120     }
4121     {
4122       \int_compare:nNnT \l_@@_final_j_int > \l_@@_col_max_int
4123       {
4124         \int_compare:nNnT { #4 } = \c_one_int
4125         { \bool_set_true:N \l_@@_final_open_bool }
4126       }
4127     }
4128   }
4129   \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.

```
4130   {
```

We do a step backwards.

```

4131         \int_sub:Nn \l_@@_final_i_int { #3 }
4132         \int_sub:Nn \l_@@_final_j_int { #4 }
4133         \bool_set_true:N \l_@@_stop_loop_bool
4134     }

```

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`.

```

4135     {
4136         \cs_if_exist:cTF
4137         {
4138             @@ _ dotted _
4139             \int_use:N \l_@@_final_i_int -
4140             \int_use:N \l_@@_final_j_int
4141         }
4142         {
4143             \int_sub:Nn \l_@@_final_i_int { #3 }
4144             \int_sub:Nn \l_@@_final_j_int { #4 }
4145             \bool_set_true:N \l_@@_final_open_bool
4146             \bool_set_true:N \l_@@_stop_loop_bool
4147         }
4148         {
4149             \cs_if_exist:cTF
4150             {
4151                 pgf @ sh @ ns @ \@@_env:
4152                 - \int_use:N \l_@@_final_i_int
4153                 - \int_use:N \l_@@_final_j_int
4154             }
4155             { \bool_set_true:N \l_@@_stop_loop_bool }
4156         }
4157     }

```

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.

```

4156     {
4157         \cs_set:cpn
4158         {
4159             @@ _ dotted _
4160             \int_use:N \l_@@_final_i_int -
4161             \int_use:N \l_@@_final_j_int
4162         }
4163         { }
4164     }
4165 }
4166 }
4167 }

```

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

```

4168     \bool_set_false:N \l_@@_stop_loop_bool
4169     \bool_do_until:Nn \l_@@_stop_loop_bool
4170     {
4171         \int_sub:Nn \l_@@_initial_i_int { #3 }
4172         \int_sub:Nn \l_@@_initial_j_int { #4 }
4173         \bool_set_false:N \l_@@_initial_open_bool
4174         \int_compare:nNnTF \l_@@_initial_i_int < \l_@@_row_min_int
4175         {
4176             \int_compare:nNnTF { #3 } = \c_one_int
4177             { \bool_set_true:N \l_@@_initial_open_bool }
4178             {
4179                 \int_compare:nNnT \l_@@_initial_j_int = { \l_@@_col_min_int - 1 }

```

```

4180         { \bool_set_true:N \l_@@_initial_open_bool }
4181     }
4182 }
4183 {
4184     \int_compare:nNnTF \l_@@_initial_j_int < \l_@@_col_min_int
4185     {
4186         \int_compare:nNnT { #4 } = \c_one_int
4187         { \bool_set_true:N \l_@@_initial_open_bool }
4188     }
4189     {
4190         \int_compare:nNnT \l_@@_initial_j_int > \l_@@_col_max_int
4191         {
4192             \int_compare:nNnT { #4 } = { -1 }
4193             { \bool_set_true:N \l_@@_initial_open_bool }
4194         }
4195     }
4196 }
4197 \bool_if:NTF \l_@@_initial_open_bool
4198 {
4199     \int_add:Nn \l_@@_initial_i_int { #3 }
4200     \int_add:Nn \l_@@_initial_j_int { #4 }
4201     \bool_set_true:N \l_@@_stop_loop_bool
4202 }
4203 {
4204     \cs_if_exist:cTF
4205     {
4206         @@ _ dotted _
4207         \int_use:N \l_@@_initial_i_int -
4208         \int_use:N \l_@@_initial_j_int
4209     }
4210     {
4211         \int_add:Nn \l_@@_initial_i_int { #3 }
4212         \int_add:Nn \l_@@_initial_j_int { #4 }
4213         \bool_set_true:N \l_@@_initial_open_bool
4214         \bool_set_true:N \l_@@_stop_loop_bool
4215     }
4216     {
4217         \cs_if_exist:cTF
4218         {
4219             pgf @ sh @ ns @ \@@_env:
4220             - \int_use:N \l_@@_initial_i_int
4221             - \int_use:N \l_@@_initial_j_int
4222         }
4223         { \bool_set_true:N \l_@@_stop_loop_bool }
4224         {
4225             \cs_set:cpn
4226             {
4227                 @@ _ dotted _
4228                 \int_use:N \l_@@_initial_i_int -
4229                 \int_use:N \l_@@_initial_j_int
4230             }
4231             { }
4232         }
4233     }
4234 }
4235 }

```

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.

```

4236     \seq_gput_right:Nx \g_@@_pos_of_xdots_seq
4237     {
4238         { \int_use:N \l_@@_initial_i_int }

```

Be careful: with `\l_@@_final_j_int` is inferior to `\l_@@_initial_j_int`. That’s why we

use `\int_min:nn` and `\int_max:nn`.

```

4239     { \int_min:nn \l_@@_initial_j_int \l_@@_final_j_int }
4240     { \int_use:N \l_@@_final_i_int }
4241     { \int_max:nn \l_@@_initial_j_int \l_@@_final_j_int }
4242     { } % for the name of the block
4243   }
4244 }

```

If the final user uses the key `xdots/shorten` in `\NiceMatrixOptions` or at the level of an environment (such as `{pNiceMatrix}`, etc.), only the so called “closed extremities” will be shortened by that key. The following command will be used *after* the detection of the extremities of a dotted line (hence at a time when we know whether the extremities are closed or open) but before the analysis of the keys of the individual command `\Cdots`, `\Vdots`. Hence, the keys `shorten`, `shorten-start` and `shorten-end` of that individual command will be applied.

```

4245 \cs_new_protected:Npn \@@_open_shorten:
4246 {
4247   \bool_if:NT \l_@@_initial_open_bool
4248   { \dim_zero:N \l_@@_xdots_shorten_start_dim }
4249   \bool_if:NT \l_@@_final_open_bool
4250   { \dim_zero:N \l_@@_xdots_shorten_end_dim }
4251 }

```

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).

```

4252 \cs_new_protected:Npn \@@_adjust_to_submatrix:nn #1 #2
4253 {
4254   \int_set:Nn \l_@@_row_min_int 1
4255   \int_set:Nn \l_@@_col_min_int 1
4256   \int_set_eq:NN \l_@@_row_max_int \c@iRow
4257   \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`.

```

4258   \seq_map_inline:Nn \g_@@_submatrix_seq
4259   { \@@_adjust_to_submatrix:nnnnnn { #1 } { #2 } ##1 }
4260 }

```

#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.

```

4261 \cs_set_protected:Npn \@@_adjust_to_submatrix:nnnnnn #1 #2 #3 #4 #5 #6
4262 {
4263   \int_compare:nNnF { #3 } > { #1 }
4264   {
4265     \int_compare:nNnF { #1 } > { #5 }
4266     {
4267       \int_compare:nNnF { #4 } > { #2 }
4268       {
4269         \int_compare:nNnF { #2 } > { #6 }
4270         {
4271           \int_set:Nn \l_@@_row_min_int
4272           { \int_max:nn \l_@@_row_min_int { #3 } }
4273           \int_set:Nn \l_@@_col_min_int
4274           { \int_max:nn \l_@@_col_min_int { #4 } }
4275           \int_set:Nn \l_@@_row_max_int
4276           { \int_min:nn \l_@@_row_max_int { #5 } }
4277           \int_set:Nn \l_@@_col_max_int
4278           { \int_min:nn \l_@@_col_max_int { #6 } }
4279         }
4280       }

```



```

4281     }
4282   }
4283 }

4284 \cs_new_protected:Npn \@@_set_initial_coords:
4285 {
4286   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4287   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4288 }
4289 \cs_new_protected:Npn \@@_set_final_coords:
4290 {
4291   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4292   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4293 }
4294 \cs_new_protected:Npn \@@_set_initial_coords_from_anchor:n #1
4295 {
4296   \pgfpointanchor
4297   {
4298     \@@_env:
4299     - \int_use:N \l_@@_initial_i_int
4300     - \int_use:N \l_@@_initial_j_int
4301   }
4302   { #1 }
4303   \@@_set_initial_coords:
4304 }
4305 \cs_new_protected:Npn \@@_set_final_coords_from_anchor:n #1
4306 {
4307   \pgfpointanchor
4308   {
4309     \@@_env:
4310     - \int_use:N \l_@@_final_i_int
4311     - \int_use:N \l_@@_final_j_int
4312   }
4313   { #1 }
4314   \@@_set_final_coords:
4315 }
4316 \cs_new_protected:Npn \@@_open_x_initial_dim:
4317 {
4318   \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
4319   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
4320   {
4321     \cs_if_exist:cT
4322     { \pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4323     {
4324       \pgfpointanchor
4325       { \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4326       { west }
4327       \dim_set:Nn \l_@@_x_initial_dim
4328       { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
4329     }
4330   }

```

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

```

4331   \dim_compare:nNtT \l_@@_x_initial_dim = \c_max_dim
4332   {
4333     \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4334     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4335     \dim_add:Nn \l_@@_x_initial_dim \col@sep
4336   }
4337 }

4338 \cs_new_protected:Npn \@@_open_x_final_dim:
4339 {
4340   \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }

```

```

4341 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
4342 {
4343   \cs_if_exist:cT
4344     { \pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4345     {
4346       \pgfpointanchor
4347         { \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4348         { east }
4349       \dim_set:Nn \l_@@_x_final_dim
4350         { \dim_max:nn \l_@@_x_final_dim \pgf@x }
4351     }
4352 }

```

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

```

4353 \dim_compare:nNnT \l_@@_x_final_dim = { - \c_max_dim }
4354 {
4355   \@@_qpoint:n { col - \int_eval:n { \l_@@_final_j_int + 1 } }
4356   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4357   \dim_sub:Nn \l_@@_x_final_dim \col@sep
4358 }
4359 }

```

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.

```

4360 \cs_new_protected:Npn \@@_draw_Ldots:nnn #1 #2 #3
4361 {
4362   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4363   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4364   {
4365     \@@_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.

```

4366   \group_begin:
4367   \@@_open_shorten:
4368   \int_if_zero:nTF { #1 }
4369     { \color { nicematrix-first-row } }
4370   {

```

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.

```

4371     \int_compare:nNnT { #1 } = \l_@@_last_row_int
4372     { \color { nicematrix-last-row } }
4373   }
4374   \keys_set:nn { NiceMatrix / xdots } { #3 }
4375   \tl_if_empty:oF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4376   \@@_actually_draw_Ldots:
4377   \group_end:
4378 }
4379 }

```

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`.

```

4380 \cs_new_protected:Npn \@@_actually_draw_Ldots:
4381 {
4382   \bool_if:NTF \l_@@_initial_open_bool
4383   {
4384     \@@_open_x_initial_dim:
4385     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4386     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4387   }
4388   { \@@_set_initial_coords_from_anchor:n { base-east } }
4389   \bool_if:NTF \l_@@_final_open_bool
4390   {
4391     \@@_open_x_final_dim:
4392     \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4393     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4394   }
4395   { \@@_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.

```

4396   \bool_lazy_all:nTF
4397   {
4398     \l_@@_initial_open_bool
4399     \l_@@_final_open_bool
4400     { \int_compare_p:nNn \l_@@_initial_i_int = \l_@@_last_row_int }
4401   }
4402   {
4403     \dim_add:Nn \l_@@_y_initial_dim \c_@@_shift_Ldots_last_row_dim
4404     \dim_add:Nn \l_@@_y_final_dim \c_@@_shift_Ldots_last_row_dim
4405   }

```

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 (?).

```

4406   {
4407     \dim_add:Nn \l_@@_y_initial_dim \l_@@_xdots_radius_dim
4408     \dim_add:Nn \l_@@_y_final_dim \l_@@_xdots_radius_dim
4409   }
4410   \@@_draw_line:
4411 }

```

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.

```

4412 \cs_new_protected:Npn \@@_draw_Cdots:nnn #1 #2 #3
4413 {
4414   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4415   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4416   {
4417     \@@_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.

```

4418   \group_begin:
4419     \@@_open_shorten:
4420     \int_if_zero:nTF { #1 }
4421     { \color { nicematrix-first-row } }
4422     {

```

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.

```

4423     \int_compare:nNnT { #1 } = \l_@@_last_row_int
4424     { \color { nicematrix-last-row } }
4425   }

```

```

4426         \keys_set:nn { NiceMatrix / xdots } { #3 }
4427         \tl_if_empty:oF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4428         \@@_actually_draw_Cdots:
4429     \group_end:
4430 }
4431 }

```

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.`

```

4432 \cs_new_protected:Npn \@@_actually_draw_Cdots:
4433 {
4434     \bool_if:NTF \l_@@_initial_open_bool
4435     { \@@_open_x_initial_dim: }
4436     { \@@_set_initial_coords_from_anchor:n { mid-east } }
4437     \bool_if:NTF \l_@@_final_open_bool
4438     { \@@_open_x_final_dim: }
4439     { \@@_set_final_coords_from_anchor:n { mid-west } }
4440     \bool_lazy_and:nnTF
4441         \l_@@_initial_open_bool
4442         \l_@@_final_open_bool
4443     {
4444         \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int }
4445         \dim_set_eq:NN \l_tmpa_dim \pgf@y
4446         \@@_qpoint:n { row - \int_eval:n { \l_@@_initial_i_int + 1 } }
4447         \dim_set:Nn \l_@@_y_initial_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
4448         \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
4449     }
4450     {
4451         \bool_if:NT \l_@@_initial_open_bool
4452         { \dim_set_eq:NN \l_@@_y_initial_dim \l_@@_y_final_dim }
4453         \bool_if:NT \l_@@_final_open_bool
4454         { \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim }
4455     }
4456     \@@_draw_line:
4457 }
4458 \cs_new_protected:Npn \@@_open_y_initial_dim:
4459 {
4460     \dim_set:Nn \l_@@_y_initial_dim { - \c_max_dim }
4461     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
4462     {
4463         \cs_if_exist:cT
4464         { pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4465         {
4466             \pgfpointanchor
4467             { \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4468             { north }
4469             \dim_set:Nn \l_@@_y_initial_dim
4470             { \dim_max:nn \l_@@_y_initial_dim \pgf@y }
4471         }
4472     }
4473     \dim_compare:nNnT \l_@@_y_initial_dim = { - \c_max_dim }
4474     {
4475         \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }

```

```

4476 \dim_set:Nn \l_@@_y_initial_dim
4477 {
4478   \fp_to_dim:n
4479   {
4480     \pgf@y
4481     + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
4482   }
4483 }
4484 }
4485 }
4486 \cs_new_protected:Npn \@@_open_y_final_dim:
4487 {
4488   \dim_set_eq:NN \l_@@_y_final_dim \c_max_dim
4489   \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
4490   {
4491     \cs_if_exist:cT
4492     { \pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4493     {
4494       \pgfpointanchor
4495       { \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4496       { south }
4497       \dim_set:Nn \l_@@_y_final_dim
4498       { \dim_min:nn \l_@@_y_final_dim \pgf@y }
4499     }
4500   }
4501   \dim_compare:nNnT \l_@@_y_final_dim = \c_max_dim
4502   {
4503     \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4504     \dim_set:Nn \l_@@_y_final_dim
4505     { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
4506   }
4507 }

```

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.

```

4508 \cs_new_protected:Npn \@@_draw_Vdots:nnn #1 #2 #3
4509 {
4510   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4511   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4512   {
4513     \@@_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.

```

4514   \group_begin:
4515   \@@_open_shorten:
4516   \int_if_zero:nTF { #2 }
4517   { \color { nicematrix-first-col } }
4518   {
4519     \int_compare:nNnT { #2 } = \l_@@_last_col_int
4520     { \color { nicematrix-last-col } }
4521   }
4522   \keys_set:nn { NiceMatrix / xdots } { #3 }
4523   \tl_if_empty:oF \l_@@_xdots_color_tl
4524   { \color { \l_@@_xdots_color_tl } }
4525   \@@_actually_draw_Vdots:
4526   \group_end:
4527 }
4528 }

```

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.

```
4529 \cs_new_protected:Npn \@@_actually_draw_Vdots:
4530 {
```

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

```
4531 \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.

```
4532 {
4533   \@@_open_y_initial_dim:
4534   \@@_open_y_final_dim:
4535   \int_if_zero:nTF \l_@@_initial_j_int
```

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

```
4536 {
4537   \@@_qpoint:n { col - 1 }
4538   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4539   \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
4540   \dim_sub:Nn \l_@@_x_initial_dim \l_@@_extra_left_margin_dim
4541   \dim_sub:Nn \l_@@_x_initial_dim \c_@@_shift_exterior_Vdots_dim
4542 }
4543 {
4544   \bool_lazy_and:nnTF
4545   { \int_compare_p:nNn \l_@@_last_col_int > { -2 } }
4546   { \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”.

```
4547 {
4548   \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4549   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4550   \dim_add:Nn \l_@@_x_initial_dim \l_@@_right_margin_dim
4551   \dim_add:Nn \l_@@_x_initial_dim \l_@@_extra_right_margin_dim
4552   \dim_add:Nn \l_@@_x_initial_dim \c_@@_shift_exterior_Vdots_dim
4553 }
```

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

```
4554 {
4555   \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4556   \dim_set_eq:NN \l_tmpa_dim \pgf@x
4557   \@@_qpoint:n { col - \int_eval:n { \l_@@_initial_j_int + 1 } }
4558   \dim_set:Nn \l_@@_x_initial_dim { ( \pgf@x + \l_tmpa_dim ) / 2 }
4559 }
4560 }
4561 }
```

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.

```
4562 {
4563   \bool_set_false:N \l_tmpa_bool
4564   \bool_if:NF \l_@@_initial_open_bool
4565   {
4566     \bool_if:NF \l_@@_final_open_bool
4567     {
4568       \@@_set_initial_coords_from_anchor:n { south-west }
4569       \@@_set_final_coords_from_anchor:n { north-west }
4570       \bool_set:Nn \l_tmpa_bool
4571       { \dim_compare_p:nNn \l_@@_x_initial_dim = \l_@@_x_final_dim }
```

```

4572     }
4573 }

```

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

```

4574 \bool_if:NTF \l_@@_initial_open_bool
4575 {
4576   \@@_open_y_initial_dim:
4577   \@@_set_final_coords_from_anchor:n { north }
4578   \dim_set_eq:NN \l_@@_x_initial_dim \l_@@_x_final_dim
4579 }
4580 {
4581   \@@_set_initial_coords_from_anchor:n { south }
4582   \bool_if:NTF \l_@@_final_open_bool
4583   \@@_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.

```

4584 {
4585   \@@_set_final_coords_from_anchor:n { north }
4586   \dim_compare:nNnF \l_@@_x_initial_dim = \l_@@_x_final_dim
4587   {
4588     \dim_set:Nn \l_@@_x_initial_dim
4589     {
4590       \bool_if:NTF \l_tmpa_bool \dim_min:nn \dim_max:nn
4591       \l_@@_x_initial_dim \l_@@_x_final_dim
4592     }
4593   }
4594 }
4595 }
4596 }
4597 \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
4598 \@@_draw_line:
4599 }

```

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.

```

4600 \cs_new_protected:Npn \@@_draw_Ddots:nnn #1 #2 #3
4601 {
4602   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4603   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4604   {
4605     \@@_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.

```

4606   \group_begin:
4607   \@@_open_shorten:
4608   \keys_set:nn { NiceMatrix / xdots } { #3 }
4609   \tl_if_empty:oF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4610   \@@_actually_draw_Ddots:
4611   \group_end:
4612 }
4613 }

```

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.

```

4614 \cs_new_protected:Npn \@@_actually_draw_Ddots:
4615 {
4616   \bool_if:NTF \l_@@_initial_open_bool
4617   {
4618     \@@_open_y_initial_dim:
4619     \@@_open_x_initial_dim:
4620   }
4621   { \@@_set_initial_coords_from_anchor:n { south~east } }
4622   \bool_if:NTF \l_@@_final_open_bool
4623   {
4624     \@@_open_x_final_dim:
4625     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4626   }
4627   { \@@_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.

```

4628   \bool_if:NT \l_@@_parallelize_diags_bool
4629   {
4630     \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).

```

4631     \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.

```

4632     {
4633       \dim_gset:Nn \g_@@_delta_x_one_dim
4634       { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4635       \dim_gset:Nn \g_@@_delta_y_one_dim
4636       { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4637     }

```

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.

```

4638     {
4639       \dim_set:Nn \l_@@_y_final_dim
4640       {
4641         \l_@@_y_initial_dim +
4642         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4643         \dim_ratio:nn \g_@@_delta_y_one_dim \g_@@_delta_x_one_dim
4644       }
4645     }
4646   }
4647   \@@_draw_line:
4648 }

```

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.

```

4649 \cs_new_protected:Npn \@@_draw_Iddots:nnn #1 #2 #3
4650 {
4651   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4652   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4653   {
4654     \@@_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.

```

4655     \group_begin:
4656     \@@_open_shorten:
4657     \keys_set:nn { NiceMatrix / xdots } { #3 }
4658     \tl_if_empty:oF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4659     \@@_actually_draw_Iddots:
4660     \group_end:
4661   }
4662 }

```

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`.

```

4663 \cs_new_protected:Npn \@@_actually_draw_Iddots:
4664 {
4665   \bool_if:NTF \l_@@_initial_open_bool
4666   {
4667     \@@_open_y_initial_dim:
4668     \@@_open_x_initial_dim:
4669   }
4670   { \@@_set_initial_coords_from_anchor:n { south-west } }
4671   \bool_if:NTF \l_@@_final_open_bool
4672   {
4673     \@@_open_y_final_dim:
4674     \@@_open_x_final_dim:
4675   }
4676   { \@@_set_final_coords_from_anchor:n { north-east } }
4677   \bool_if:NT \l_@@_parallelize_diags_bool
4678   {
4679     \int_gincr:N \g_@@_iddots_int
4680     \int_compare:nNnTF \g_@@_iddots_int = \c_one_int
4681     {
4682       \dim_gset:Nn \g_@@_delta_x_two_dim
4683       { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4684       \dim_gset:Nn \g_@@_delta_y_two_dim
4685       { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4686     }
4687     {
4688       \dim_set:Nn \l_@@_y_final_dim
4689       {
4690         \l_@@_y_initial_dim +
4691         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4692         \dim_ratio:nn \g_@@_delta_y_two_dim \g_@@_delta_x_two_dim
4693       }
4694     }
4695   }
4696   \@@_draw_line:
4697 }

```

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`

```

4698 \cs_new_protected:Npn \@@_draw_line:
4699 {
4700   \pgfrememberpicturepositiononpagetrue
4701   \pgf@relevantforpicturesizefalse
4702   \bool_lazy_or:nnTF
4703     { \tl_if_eq_p:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl }
4704     \l_@@_dotted_bool
4705     \@@_draw_standard_dotted_line:
4706     \@@_draw_unstandard_dotted_line:
4707 }

```

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.

```

4708 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:
4709 {
4710   \begin { scope }
4711     \@@_draw_unstandard_dotted_line:o
4712     { \l_@@_xdots_line_style_tl , \l_@@_xdots_color_tl }
4713 }

```

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.

```

4714 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:n #1
4715 {
4716   \@@_draw_unstandard_dotted_line:nooo
4717   { #1 }
4718   \l_@@_xdots_up_tl
4719   \l_@@_xdots_down_tl
4720   \l_@@_xdots_middle_tl
4721 }
4722 \cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:n { o }

```

The following Tikz styles are for the three labels (set by the symbols `_`, `^` and `=`) of a continuous line with a non-standard style.

```

4723 \hook_gput_code:nnn { begindocument } { . }
4724 {
4725   \IfPackageLoadedTF { tikz }
4726   {
4727     \tikzset
4728     {
4729       @@_node_above / .style = { sloped , above } ,
4730       @@_node_below / .style = { sloped , below } ,
4731       @@_node_middle / .style =
4732       {

```

```

4733         sloped ,
4734         inner~sep = \c_@@_innersep_middle_dim
4735     }
4736 }
4737 }
4738 { }
4739 }

```

```

4740 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:nnnn #1 #2 #3 #4
4741 {

```

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.

```

4742     \dim_zero_new:N \l_@@_l_dim
4743     \dim_set:Nn \l_@@_l_dim
4744     {
4745         \fp_to_dim:n
4746         {
4747             sqrt
4748             (
4749                 ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
4750                 +
4751                 ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
4752             )
4753         }
4754     }

```

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.

```

4755     \dim_compare:nNnT \l_@@_l_dim < \c_@@_max_l_dim
4756     {
4757         \dim_compare:nNnT \l_@@_l_dim > { 1 pt }
4758         \@@_draw_unstandard_dotted_line_i:
4759     }

```

If the key `xdots/horizontal-labels` has been used.

```

4760     \bool_if:NT \l_@@_xdots_h_labels_bool
4761     {
4762         \tikzset
4763         {
4764             @@_node_above / .style = { auto = left } ,
4765             @@_node_below / .style = { auto = right } ,
4766             @@_node_middle / .style = { inner-sep = \c_@@_innersep_middle_dim }
4767         }
4768     }
4769     \tl_if_empty:nF { #4 }
4770     { \tikzset { @@_node_middle / .append~style = { fill = white } } }
4771     \draw
4772     [ #1 ]
4773     ( \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).

```

4774         -- node [ @@_node_middle ] { $ \scriptstyle #4 $ }
4775         node [ @@_node_below ] { $ \scriptstyle #3 $ }
4776         node [ @@_node_above ] { $ \scriptstyle #2 $ }
4777         ( \l_@@_x_final_dim , \l_@@_y_final_dim ) ;
4778     \end { scope }
4779 }

```

```

4780 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line_i:
4781 {
4782   \dim_set:Nn \l_tmpa_dim
4783   {
4784     \l_@@_x_initial_dim
4785     + ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
4786     * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
4787   }
4788   \dim_set:Nn \l_tmpb_dim
4789   {
4790     \l_@@_y_initial_dim
4791     + ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
4792     * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
4793   }
4794   \dim_set:Nn \l_@@_tmpc_dim
4795   {
4796     \l_@@_x_final_dim
4797     - ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
4798     * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
4799   }
4800   \dim_set:Nn \l_@@_tmpd_dim
4801   {
4802     \l_@@_y_final_dim
4803     - ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
4804     * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
4805   }
4806   \dim_set_eq:NN \l_@@_x_initial_dim \l_tmpa_dim
4807   \dim_set_eq:NN \l_@@_y_initial_dim \l_tmpb_dim
4808   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_tmpc_dim
4809   \dim_set_eq:NN \l_@@_y_final_dim \l_@@_tmpd_dim
4810 }
4811 \cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:nnnn { n o o o }

```

The command `\@@_draw_standard_dotted_line:` draws the line with our system of dots (which gives a dotted line with real rounded dots).

```

4812 \cs_new_protected:Npn \@@_draw_standard_dotted_line:
4813 {
4814   \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.

```

4815   \dim_zero_new:N \l_@@_l_dim
4816   \dim_set:Nn \l_@@_l_dim
4817   {
4818     \fp_to_dim:n
4819     {
4820       sqrt
4821       (
4822         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
4823         +
4824         ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
4825       )
4826     }
4827   }

```

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.

```

4828   \dim_compare:nNnT \l_@@_l_dim < \c_@@_max_l_dim
4829   {
4830     \dim_compare:nNnT \l_@@_l_dim > { 1 pt }
4831     \@@_draw_standard_dotted_line_i:

```

```

4832     }
4833 \group_end:
4834 \bool_lazy_all:nF
4835 {
4836   { \tl_if_empty_p:N \l_@@_xdots_up_tl }
4837   { \tl_if_empty_p:N \l_@@_xdots_down_tl }
4838   { \tl_if_empty_p:N \l_@@_xdots_middle_tl }
4839 }
4840 \l_@@_labels_standard_dotted_line:
4841 }
4842 \dim_const:Nn \c_@@_max_l_dim { 50 cm }
4843 \cs_new_protected:Npn \@@_draw_standard_dotted_line_i:
4844 {

```

The number of dots will be $\l_tmpa_int + 1$.

```

4845   \int_set:Nn \l_tmpa_int
4846   {
4847     \dim_ratio:nn
4848     {
4849       \l_@@_l_dim
4850       - \l_@@_xdots_shorten_start_dim
4851       - \l_@@_xdots_shorten_end_dim
4852     }
4853     \l_@@_xdots_inter_dim
4854   }

```

The dimensions \l_tmpa_dim and \l_tmpb_dim are the coordinates of the vector between two dots in the dotted line.

```

4855   \dim_set:Nn \l_tmpa_dim
4856   {
4857     ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4858     \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
4859   }
4860   \dim_set:Nn \l_tmpb_dim
4861   {
4862     ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
4863     \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
4864   }

```

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.

```

4865   \dim_gadd:Nn \l_@@_x_initial_dim
4866   {
4867     ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4868     \dim_ratio:nn
4869     {
4870       \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
4871       + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
4872     }
4873     { 2 \l_@@_l_dim }
4874   }
4875   \dim_gadd:Nn \l_@@_y_initial_dim
4876   {
4877     ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
4878     \dim_ratio:nn
4879     {
4880       \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
4881       + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
4882     }
4883     { 2 \l_@@_l_dim }
4884   }
4885   \pgf@relevantforpicturesizefalse

```

```

4886 \int_step_inline:nnn \c_zero_int \l_tmpa_int
4887 {
4888   \pgfpathcircle
4889   { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
4890   { \l_@@_xdots_radius_dim }
4891   \dim_add:Nn \l_@@_x_initial_dim \l_tmpa_dim
4892   \dim_add:Nn \l_@@_y_initial_dim \l_tmpb_dim
4893 }
4894 \pgfusepathqfill
4895 }

4896 \cs_new_protected:Npn \l_@@_labels_standard_dotted_line:
4897 {
4898   \pgfscope
4899   \pgftransformshift
4900   {
4901     \pgfpointlineattime { 0.5 }
4902     { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
4903     { \pgfpoint \l_@@_x_final_dim \l_@@_y_final_dim }
4904   }
4905   \fp_set:Nn \l_tmpa_fp
4906   {
4907     atand
4908     (
4909       \l_@@_y_final_dim - \l_@@_y_initial_dim ,
4910       \l_@@_x_final_dim - \l_@@_x_initial_dim
4911     )
4912   }
4913   \pgftransformrotate { \fp_use:N \l_tmpa_fp }
4914   \bool_if:NF \l_@@_xdots_h_labels_bool { \fp_zero:N \l_tmpa_fp }
4915   \tl_if_empty:NF \l_@@_xdots_middle_tl
4916   {
4917     \begin { pgfscope }
4918     \pgfset { inner~sep = \c_@@_innersep_middle_dim }
4919     \pgfnode
4920     { rectangle }
4921     { center }
4922     {
4923       \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4924       {
4925         \c_math_toggle_token
4926         \scriptstyle \l_@@_xdots_middle_tl
4927         \c_math_toggle_token
4928       }
4929     }
4930     { }
4931     {
4932       \pgfsetfillcolor { white }
4933       \pgfusepath { fill }
4934     }
4935     \end { pgfscope }
4936   }
4937   \tl_if_empty:NF \l_@@_xdots_up_tl
4938   {
4939     \pgfnode
4940     { rectangle }
4941     { south }
4942     {
4943       \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4944       {
4945         \c_math_toggle_token
4946         \scriptstyle \l_@@_xdots_up_tl
4947         \c_math_toggle_token

```

```

4948         }
4949     }
4950     { }
4951     { \pgfusepath { } }
4952 }
4953 \tl_if_empty:NF \l_@@_xdots_down_tl
4954 {
4955     \pgfnode
4956     { rectangle }
4957     { north }
4958     {
4959         \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4960         {
4961             \c_math_toggle_token
4962             \scriptstyle \l_@@_xdots_down_tl
4963             \c_math_toggle_token
4964         }
4965     }
4966     { }
4967     { \pgfusepath { } }
4968 }
4969 \endpgfscope
4970 }

```

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.

```

4971 \hook_gput_code:nnn { begindocument } { . }
4972 {
4973     \cs_set_nopar:Npn \l_@@_argspec_tl { m E { _ ^ : } { { } { } { } } }
4974     \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
4975     \cs_new_protected:Npn \@@_Ldots
4976     { \@@_collect_options:n { \@@_Ldots_i } }
4977     \exp_args:NNo \NewDocumentCommand \@@_Ldots_i \l_@@_argspec_tl
4978     {
4979         \int_if_zero:nTF \c@jCol
4980         { \@@_error:nn { in~first~col } \Ldots }
4981         {
4982             \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
4983             { \@@_error:nn { in~last~col } \Ldots }
4984             {
4985                 \@@_instruction_of_type:nnn \c_false_bool { Ldots }
4986                 { #1 , down = #2 , up = #3 , middle = #4 }
4987             }
4988         }
4989         \bool_if:NF \l_@@_nullify_dots_bool
4990         { \phantom { \ensuremath { \@@_old_ldots } } }
4991         \bool_gset_true:N \g_@@_empty_cell_bool
4992     }

```

```

4993 \cs_new_protected:Npn \@@_Cdots
4994 { \@@_collect_options:n { \@@_Cdots_i } }
4995 \exp_args:NNo \NewDocumentCommand \@@_Cdots_i \l_@@_argspec_tl
4996 {
4997   \int_if_zero:nTF \c@jCol
4998   { \@@_error:nn { in~first~col } \Cdots }
4999   {
5000     \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
5001     { \@@_error:nn { in~last~col } \Cdots }
5002     {
5003       \@@_instruction_of_type:nnn \c_false_bool { Cdots }
5004       { #1 , down = #2 , up = #3 , middle = #4 }
5005     }
5006   }
5007   \bool_if:NF \l_@@_nullify_dots_bool
5008   { \phantom { \ensuremath { \@@_old_cdots } } }
5009   \bool_gset_true:N \g_@@_empty_cell_bool
5010 }

5011 \cs_new_protected:Npn \@@_Vdots
5012 { \@@_collect_options:n { \@@_Vdots_i } }
5013 \exp_args:NNo \NewDocumentCommand \@@_Vdots_i \l_@@_argspec_tl
5014 {
5015   \int_if_zero:nTF \c@iRow
5016   { \@@_error:nn { in~first~row } \Vdots }
5017   {
5018     \int_compare:nNnTF \c@iRow = \l_@@_last_row_int
5019     { \@@_error:nn { in~last~row } \Vdots }
5020     {
5021       \@@_instruction_of_type:nnn \c_false_bool { Vdots }
5022       { #1 , down = #2 , up = #3 , middle = #4 }
5023     }
5024   }
5025   \bool_if:NF \l_@@_nullify_dots_bool
5026   { \phantom { \ensuremath { \@@_old_vdots } } }
5027   \bool_gset_true:N \g_@@_empty_cell_bool
5028 }

5029 \cs_new_protected:Npn \@@_Ddots
5030 { \@@_collect_options:n { \@@_Ddots_i } }
5031 \exp_args:NNo \NewDocumentCommand \@@_Ddots_i \l_@@_argspec_tl
5032 {
5033   \int_case:nnF \c@iRow
5034   {
5035     0 { \@@_error:nn { in~first~row } \Ddots }
5036     \l_@@_last_row_int { \@@_error:nn { in~last~row } \Ddots }
5037   }
5038   {
5039     \int_case:nnF \c@jCol
5040     {
5041       0 { \@@_error:nn { in~first~col } \Ddots }
5042       \l_@@_last_col_int { \@@_error:nn { in~last~col } \Ddots }
5043     }
5044     {
5045       \keys_set_known:nn { NiceMatrix / Ddots } { #1 }
5046       \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Ddots }
5047       { #1 , down = #2 , up = #3 , middle = #4 }
5048     }
5049   }
5050   \bool_if:NF \l_@@_nullify_dots_bool
5051   { \phantom { \ensuremath { \@@_old_ddots } } }

```



```

5053     \bool_gset_true:N \g_@@_empty_cell_bool
5054 }

5055 \cs_new_protected:Npn \@@_Iddots
5056 { \@@_collect_options:n { \@@_Iddots_i } }
5057 \exp_args:NNo \NewDocumentCommand \@@_Iddots_i \l_@@_argspec_tl
5058 {
5059     \int_case:nnF \c@iRow
5060     {
5061         0 { \@@_error:nn { in~first~row } \Iddots }
5062         \l_@@_last_row_int { \@@_error:nn { in~last~row } \Iddots }
5063     }
5064     {
5065         \int_case:nnF \c@jCol
5066         {
5067             0 { \@@_error:nn { in~first~col } \Iddots }
5068             \l_@@_last_col_int { \@@_error:nn { in~last~col } \Iddots }
5069         }
5070         {
5071             \keys_set_known:nn { NiceMatrix / Ddots } { #1 }
5072             \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Iddots }
5073             { #1 , down = #2 , up = #3 , middle = #4 }
5074         }
5075     }
5076     \bool_if:NF \l_@@_nullify_dots_bool
5077     { \phantom { \ensuremath { \@@_old_iddots } } }
5078     \bool_gset_true:N \g_@@_empty_cell_bool
5079 }
5080 }

```

End of the \AddToHook.

Despite its name, the following set of keys will be used for \Ddots but also for \Iddots.

```

5081 \keys_define:nn { NiceMatrix / Ddots }
5082 {
5083     draw-first .bool_set:N = \l_@@_draw_first_bool ,
5084     draw-first .default:n = true ,
5085     draw-first .value_forbidden:n = true
5086 }

```

The command \@@_Hspace: will be linked to \hspace in {NiceArray}.

```

5087 \cs_new_protected:Npn \@@_Hspace:
5088 {
5089     \bool_gset_true:N \g_@@_empty_cell_bool
5090     \hspace
5091 }

```

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`.

```

5092 \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`.

```

5093 \cs_new:Npn \@@_Hdotsfor:
5094 {
5095     \bool_lazy_and:nnTF
5096     { \int_if_zero_p:n \c@jCol }
5097     { \int_if_zero_p:n \l_@@_first_col_int }
5098     {

```

```

5099     \bool_if:NTF \g_@@_after_col_zero_bool
5100     {
5101         \multicolumn { 1 } { c } { }
5102         \@@_Hdotsfor_i
5103     }
5104     { \@@_fatal:n { Hdotsfor~in~col~0 } }
5105 }
5106 {
5107     \multicolumn { 1 } { c } { }
5108     \@@_Hdotsfor_i
5109 }
5110 }

```

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:`).

```

5111 \hook_gput_code:nnn { begindocument } { . }
5112 {
5113     \cs_set_nopar:Npn \l_@@_argspec_tl { m m 0 { } E { _ ^ : } { { } { } { } } }
5114     \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.

```

5115     \cs_new_protected:Npn \@@_Hdotsfor_i
5116     { \@@_collect_options:n { \@@_Hdotsfor_ii } }
5117     \exp_args:NNo \NewDocumentCommand \@@_Hdotsfor_ii \l_@@_argspec_tl
5118     {
5119         \tl_gput_right:Nx \g_@@_HVdotsfor_lines_tl
5120         {
5121             \@@_Hdotsfor:nnnn
5122             { \int_use:N \c@iRow }
5123             { \int_use:N \c@jCol }
5124             { #2 }
5125             {
5126                 #1 , #3 ,
5127                 down = \exp_not:n { #4 } ,
5128                 up = \exp_not:n { #5 } ,
5129                 middle = \exp_not:n { #6 }
5130             }
5131         }
5132         \prg_replicate:nn { #2 - 1 }
5133         {
5134             &
5135             \multicolumn { 1 } { c } { }
5136             \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
5137         }
5138     }
5139 }

```

```

5140 \cs_new_protected:Npn \@@_Hdotsfor:nnnn #1 #2 #3 #4
5141 {
5142     \bool_set_false:N \l_@@_initial_open_bool
5143     \bool_set_false:N \l_@@_final_open_bool

```

For the row, it's easy.

```

5144     \int_set:Nn \l_@@_initial_i_int { #1 }
5145     \int_set_eq:NN \l_@@_final_i_int \l_@@_initial_i_int

```

For the column, it's a bit more complicated.

```

5146     \int_compare:nNnTF { #2 } = \c_one_int
5147     {
5148         \int_set_eq:NN \l_@@_initial_j_int \c_one_int
5149         \bool_set_true:N \l_@@_initial_open_bool
5150     }
5151     {

```

```

5152     \cs_if_exist:cTF
5153     {
5154         pgf @ sh @ ns @ \@@_env:
5155         - \int_use:N \l_@@_initial_i_int
5156         - \int_eval:n { #2 - 1 }
5157     }
5158     { \int_set:Nn \l_@@_initial_j_int { #2 - 1 } }
5159     {
5160         \int_set:Nn \l_@@_initial_j_int { #2 }
5161         \bool_set_true:N \l_@@_initial_open_bool
5162     }
5163 }
5164 \int_compare:nNnTF { #2 + #3 - 1 } = \c@jCol
5165 {
5166     \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5167     \bool_set_true:N \l_@@_final_open_bool
5168 }
5169 {
5170     \cs_if_exist:cTF
5171     {
5172         pgf @ sh @ ns @ \@@_env:
5173         - \int_use:N \l_@@_final_i_int
5174         - \int_eval:n { #2 + #3 }
5175     }
5176     { \int_set:Nn \l_@@_final_j_int { #2 + #3 } }
5177     {
5178         \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5179         \bool_set_true:N \l_@@_final_open_bool
5180     }
5181 }
5182 \group_begin:
5183 \@@_open_shorten:
5184 \int_if_zero:nTF { #1 }
5185 { \color { nicematrix-first-row } }
5186 {
5187     \int_compare:nNnT { #1 } = \g_@@_row_total_int
5188     { \color { nicematrix-last-row } }
5189 }
5190
5191 \keys_set:nn { NiceMatrix / xdots } { #4 }
5192 \tl_if_empty:oF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
5193 \@@_actually_draw_ldots:
5194 \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).

5195     \int_step_inline:nnn { #2 } { #2 + #3 - 1 }
5196     { \cs_set:cpn { @@ _ dotted _ #1 - ##1 } { } }
5197 }

5198 \hook_gput_code:nnn { begindocument } { . }
5199 {
5200     \cs_set_nopar:Npn \l_@@_argspec_tl { m m 0 { } E { _ ^ : } { { } { } { } } }
5201     \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
5202     \cs_new_protected:Npn \@@_Vdotsfor:
5203     { \@@_collect_options:n { \@@_Vdotsfor_i } }
5204     \exp_args:NNo \NewDocumentCommand \@@_Vdotsfor_i \l_@@_argspec_tl
5205     {
5206         \bool_gset_true:N \g_@@_empty_cell_bool
5207         \tl_gput_right:Nx \g_@@_HVdotsfor_lines_tl
5208         {

```

```

5209 \@@_Vdotsfor:nnnn
5210 { \int_use:N \c@iRow }
5211 { \int_use:N \c@jCol }
5212 { #2 }
5213 {
5214   #1 , #3 ,
5215   down = \exp_not:n { #4 } ,
5216   up = \exp_not:n { #5 } ,
5217   middle = \exp_not:n { #6 }
5218 }
5219 }
5220 }
5221 }

```

```

5222 \cs_new_protected:Npn \@@_Vdotsfor:nnnn #1 #2 #3 #4
5223 {
5224   \bool_set_false:N \l_@@_initial_open_bool
5225   \bool_set_false:N \l_@@_final_open_bool

```

For the column, it's easy.

```

5226 \int_set:Nn \l_@@_initial_j_int { #2 }
5227 \int_set_eq:NN \l_@@_final_j_int \l_@@_initial_j_int

```

For the row, it's a bit more complicated.

```

5228 \int_compare:nNnTF { #1 } = \c_one_int
5229 {
5230   \int_set_eq:NN \l_@@_initial_i_int \c_one_int
5231   \bool_set_true:N \l_@@_initial_open_bool
5232 }
5233 {
5234   \cs_if_exist:cTF
5235   {
5236     pgf @ sh @ ns @ \@@_env:
5237     - \int_eval:n { #1 - 1 }
5238     - \int_use:N \l_@@_initial_j_int
5239   }
5240   { \int_set:Nn \l_@@_initial_i_int { #1 - 1 } }
5241   {
5242     \int_set:Nn \l_@@_initial_i_int { #1 }
5243     \bool_set_true:N \l_@@_initial_open_bool
5244   }
5245 }
5246 \int_compare:nNnTF { #1 + #3 - 1 } = \c@iRow
5247 {
5248   \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5249   \bool_set_true:N \l_@@_final_open_bool
5250 }
5251 {
5252   \cs_if_exist:cTF
5253   {
5254     pgf @ sh @ ns @ \@@_env:
5255     - \int_eval:n { #1 + #3 }
5256     - \int_use:N \l_@@_final_j_int
5257   }
5258   { \int_set:Nn \l_@@_final_i_int { #1 + #3 } }
5259   {
5260     \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5261     \bool_set_true:N \l_@@_final_open_bool
5262   }
5263 }
5264 \group_begin:
5265 \@@_open_shorten:
5266 \int_if_zero:nTF { #2 }

```

```

5267     { \color { nicematrix-first-col } }
5268     {
5269         \int_compare:nNt { #2 } = \g_@@_col_total_int
5270         { \color { nicematrix-last-col } }
5271     }
5272     \keys_set:nn { NiceMatrix / xdots } { #4 }
5273     \tl_if_empty:oF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
5274     \@@_actually_draw_Vdots:
5275     \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).

```

5276     \int_step_inline:nnn { #1 } { #1 + #3 - 1 }
5277     { \cs_set:cpn { @@ _ dotted _ ##1 - #2 } { } }
5278 }

```

The command `\@@_rotate:` will be linked to `\rotate` in `{NiceArrayWithDelims}`.

```

5279 \NewDocumentCommand \@@_rotate: { 0 { } }
5280 {
5281     \peek_remove_spaces:n
5282     {
5283         \bool_gset_true:N \g_@@_rotate_bool
5284         \keys_set:nn { NiceMatrix / rotate } { #1 }
5285     }
5286 }

5287 \keys_define:nn { NiceMatrix / rotate }
5288 {
5289     c .code:n = \bool_gset_true:N \g_@@_rotate_c_bool ,
5290     c .value_forbidden:n = true ,
5291     unknown .code:n = \@@_error:n { Unknown~key~for~rotate }
5292 }

```

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

```

5293 \cs_new:Npn \@@_double_int_eval:n #1-#2 \q_stop
5294 {
5295     \tl_if_empty:nTF { #2 }
5296     { #1 }

```

¹³Indeed, we want that the user may use the command `\line` in `\CodeAfter` with LaTeX counters in the arguments — with the command `\value`.

```

5297     { \@@_double_int_eval_i:n #1-#2 \q_stop }
5298   }
5299   \cs_new:Npn \@@_double_int_eval_i:n #1-#2- \q_stop
5300   { \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).

```

5301   \hook_gput_code:nnn { begindocument } { . }
5302   {
5303     \cs_set_nopar:Npn \l_@@_argspec_tl
5304     { 0 { } m m ! 0 { } E { _ ^ : } { { } { } { } { } } }
5305     \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
5306     \exp_args:NNo \NewDocumentCommand \@@_line \l_@@_argspec_tl
5307     {
5308       \group_begin:
5309       \keys_set:nn { NiceMatrix / xdots } { #1 , #4 , down = #5 , up = #6 }
5310       \tl_if_empty:oF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
5311       \use:e
5312       {
5313         \@@_line_i:nn
5314         { \@@_double_int_eval:n #2 - \q_stop }
5315         { \@@_double_int_eval:n #3 - \q_stop }
5316       }
5317       \group_end:
5318     }
5319   }

5320   \cs_new_protected:Npn \@@_line_i:nn #1 #2
5321   {
5322     \bool_set_false:N \l_@@_initial_open_bool
5323     \bool_set_false:N \l_@@_final_open_bool
5324     \bool_lazy_or:nnTF
5325     { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #1 } }
5326     { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #2 } }
5327     { \@@_error:nnn { unknown-cell-for-line-in-CodeAfter } { #1 } { #2 } }

```

The test of `measuring@` is a security (cf. question 686649 on TeX StackExchange).

```

5328     { \legacy_if:nF { measuring@ } { \@@_draw_line_ii:nn { #1 } { #2 } } }
5329   }

5330   \hook_gput_code:nnn { begindocument } { . }
5331   {
5332     \cs_new_protected:Npx \@@_draw_line_ii:nn #1 #2
5333     {

```

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:`.

```

5334     \c_@@_pgfortikzpicture_tl
5335     \@@_draw_line_iii:nn { #1 } { #2 }
5336     \c_@@_endpgfortikzpicture_tl
5337   }
5338 }

```

The following command *must* be protected (it’s used in the construction of `\@@_draw_line_ii:nn`).

```

5339   \cs_new_protected:Npn \@@_draw_line_iii:nn #1 #2
5340   {
5341     \pgfrememberpicturepositiononpagetrue
5342     \pgfpointshapeborder { \@@_env: - #1 } { \@@_qpoint:n { #2 } }
5343     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
5344     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
5345     \pgfpointshapeborder { \@@_env: - #2 } { \@@_qpoint:n { #1 } }
5346     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x

```

```

5347 \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
5348 \@@_draw_line:
5349 }

```

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_than:nn` ensures that the instructions are inserted only if you are in a row which is (still) in the scope of that instructions (which depends on the value of the key `nb-rows` of `\RowStyle`).

That test will be active even in an expandable context because `\@@_if_row_less_than:nn` is *not* protected.

`#1` is the first row *after* the scope of the instructions in `#2`

```

5350 \cs_new:Npn \@@_if_row_less_than:nn #1 #2
5351 { \int_compare:nNnT { \c@iRow } < { #1 } { #2 } }

```

`\@@_put_in_row_style` will be used several times by `\RowStyle`.

```

5352 \cs_set_protected:Npn \@@_put_in_row_style:n #1
5353 {
5354   \tl_gput_right:Nx \g_@@_row_style_tl
5355   {

```

Be careful, `\exp_not:N \@@_if_row_less_than:nn` can't be replaced by a protected version of `\@@_if_row_less_than:nn`.

```

5356 \exp_not:N
5357 \@@_if_row_less_than:nn
5358 { \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`).

```

5359 { \exp_not:n { #1 } \scan_stop: }
5360 }
5361 }
5362 \cs_generate_variant:Nn \@@_put_in_row_style:n { e }

```

```

5363 \keys_define:nn { NiceMatrix / RowStyle }
5364 {
5365   cell-space-top-limit .dim_set:N = \l_tmpa_dim ,
5366   cell-space-top-limit .value_required:n = true ,
5367   cell-space-bottom-limit .dim_set:N = \l_tmpb_dim ,
5368   cell-space-bottom-limit .value_required:n = true ,
5369   cell-space-limits .meta:n =
5370   {
5371     cell-space-top-limit = #1 ,
5372     cell-space-bottom-limit = #1 ,
5373   } ,
5374   color .tl_set:N = \l_@@_color_tl ,
5375   color .value_required:n = true ,
5376   bold .bool_set:N = \l_@@_bold_row_style_bool ,
5377   bold .default:n = true ,
5378   nb-rows .code:n =
5379   \str_if_eq:nnTF { #1 } { * }

```

```

5380     { \int_set:Nn \l_@@_key_nb_rows_int { 500 } }
5381     { \int_set:Nn \l_@@_key_nb_rows_int { #1 } } ,
5382     nb-rows .value_required:n = true ,
5383     rowcolor .tl_set:N = \l_tmpa_tl ,
5384     rowcolor .value_required:n = true ,
5385     unknown .code:n = \@@_error:n { Unknown~key~for~RowStyle }
5386 }

```

```

5387 \NewDocumentCommand \@@_RowStyle:n { 0 { } m }
5388 {
5389   \group_begin:
5390   \tl_clear:N \l_tmpa_tl
5391   \tl_clear:N \l_@@_color_tl
5392   \int_set_eq:NN \l_@@_key_nb_rows_int \c_one_int
5393   \dim_zero:N \l_tmpa_dim
5394   \dim_zero:N \l_tmpb_dim
5395   \keys_set:nm { NiceMatrix / RowStyle } { #1 }

```

If the key rowcolor has been used.

```

5396   \tl_if_empty:NF \l_tmpa_tl
5397   {

```

First, the end of the current row (we remind that \RowStyle applies to the *end* of the current row).

```

5398     \tl_gput_right:Nx \g_@@_pre_code_before_tl
5399     {

```

The command \@@_exp_color_arg:No is *fully expandable*.

```

5400         \@@_exp_color_arg:No \@@_rectanglecolor \l_tmpa_tl
5401         { \int_use:N \c@iRow - \int_use:N \c@jCol }
5402         { \int_use:N \c@iRow - * }
5403     }

```

Then, the other rows (if there is several rows).

```

5404     \int_compare:nNnT \l_@@_key_nb_rows_int > \c_one_int
5405     {
5406       \tl_gput_right:Nx \g_@@_pre_code_before_tl
5407       {
5408         \@@_exp_color_arg:No \@@_rowcolor \l_tmpa_tl
5409         {
5410           \int_eval:n { \c@iRow + 1 }
5411           - \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int - 1 }
5412         }
5413       }
5414     }
5415   }
5416   \@@_put_in_row_style:n { \exp_not:n { #2 } }

```

\l_tmpa_dim is the value of the key cell-space-top-limit of \RowStyle.

```

5417   \dim_compare:nNnT \l_tmpa_dim > \c_zero_dim
5418   {
5419     \exp_args:Nx \@@_put_in_row_style:n
5420     {
5421       \tl_gput_right:Nn \exp_not:N \g_@@_cell_after_hook_tl
5422       {

```

It's not possible to change the following code by using \dim_set_eq:NN (because of expansion).

```

5423         \dim_set:Nn \l_@@_cell_space_top_limit_dim
5424         { \dim_use:N \l_tmpa_dim }
5425     }
5426   }
5427 }

```


`\l_tmpb_dim` is the value of the key `cell-space-bottom-limit` of `\RowStyle`.

```

5428   \dim_compare:nNnT \l_tmpb_dim > \c_zero_dim
5429   {
5430     \exp_args:Nx \@@_put_in_row_style:n
5431     {
5432       \tl_gput_right:Nn \exp_not:N \g_@@_cell_after_hook_tl
5433       {
5434         \dim_set:Nn \l_@@_cell_space_bottom_limit_dim
5435         { \dim_use:N \l_tmpb_dim }
5436       }
5437     }
5438   }

```

`\l_@@_color_tl` is the value of the key `color` of `\RowStyle`.

```

5439   \tl_if_empty:NF \l_@@_color_tl
5440   {
5441     \@@_put_in_row_style:e
5442     {
5443       \mode_leave_vertical:
5444       \@@_color:n { \l_@@_color_tl }
5445     }
5446   }

```

`\l_@@_bold_row_style_bool` is the value of the key `bold`.

```

5447   \bool_if:NT \l_@@_bold_row_style_bool
5448   {
5449     \@@_put_in_row_style:n
5450     {
5451       \exp_not:n
5452       {
5453         \if_mode_math:
5454         \c_math_toggle_token
5455         \bfseries \boldmath
5456         \c_math_toggle_token
5457       }
5458       \else:
5459       \bfseries \boldmath
5460       \fi:
5461     }
5462   }
5463   \group_end:
5464   \g_@@_row_style_tl
5465   \ignorespaces
5466 }

```

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).

```
5467 \cs_new_protected:Npn \@@_add_to_colors_seq:nn #1 #2
5468 {
```

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.

```
5469 \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`.

```
5470 \str_if_in:nnF { #1 } { !! }
5471 {
5472   \seq_map_indexed_inline:Nn \g_@@_colors_seq
5473   { \tl_if_eq:nnT { #1 } { ##2 } { \int_set:Nn \l_tmpa_int { ##2 } } }
5474 }
5475 \int_if_zero:nTF \l_tmpa_int
```

First, the case where the color is a *new* color (not in the sequence).

```
5476 {
5477   \seq_gput_right:Nn \g_@@_colors_seq { #1 }
5478   \tl_gset:cx { g_@@_color _ \seq_count:N \g_@@_colors_seq _ tl } { #2 }
5479 }
```

Now, the case where the color is *not* a new color (the color is in the sequence at the position `\l_tmpa_int`).

```
5480 { \tl_gput_right:cx { g_@@_color _ \int_use:N \l_tmpa_int _tl } { #2 } }
5481 }
5482 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e n }
5483 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e e }
```

The following command must be used within a `\pgfpicture`.

```
5484 \cs_new_protected:Npn \@@_clip_with_rounded_corners:
5485 {
5486   \dim_compare:nNnT \l_@@_tab_rounded_corners_dim > \c_zero_dim
5487   {
```

The TeX group is for `\pgfsetcornersarced` (whose scope is the TeX scope).

```
5488 \group_begin:
5489 \pgfsetcornersarced
5490 {
5491   \pgfpoint
5492   { \l_@@_tab_rounded_corners_dim }
5493   { \l_@@_tab_rounded_corners_dim }
5494 }
```

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.

```
5495 \bool_if:NTF \l_@@_hvlines_bool
5496 {
5497   \pgfpathrectanglecorners
5498   {
```

```

5499         \pgfpointadd
5500         { \@@_qpoint:n { row-1 } }
5501         { \pgfpoint { 0.5 \arrayrulewidth } { \c_zero_dim } }
5502     }
5503     {
5504         \pgfpointadd
5505         {
5506             \@@_qpoint:n
5507             { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
5508         }
5509         { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
5510     }
5511 }
5512 {
5513     \pgfpathrectanglecorners
5514     { \@@_qpoint:n { row-1 } }
5515     {
5516         \pgfpointadd
5517         {
5518             \@@_qpoint:n
5519             { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
5520         }
5521         { \pgfpoint \c_zero_dim \arrayrulewidth }
5522     }
5523 }
5524 \pgfusepath { clip }
5525 \group_end:

```

The TeX group was for \pgfsetcornersarced.

```

5526     }
5527 }

```

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`).

```

5528 \cs_new_protected:Npn \@@_actually_color:
5529 {
5530     \pgfpicture
5531     \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.

```

5532     \@@_clip_with_rounded_corners:
5533     \seq_map_indexed_inline:Nn \g_@@_colors_seq
5534     {
5535         \int_compare:nNnTF { ##1 } = \c_one_int
5536         {
5537             \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_nocolor:n
5538             \use:c { g_@@_color _ 1 _tl }
5539             \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_normal:n
5540         }
5541         {
5542             \begin { pgfscope }
5543                 \@@_color_opacity ##2
5544                 \use:c { g_@@_color _ ##1 _tl }
5545                 \tl_gclear:c { g_@@_color _ ##1 _tl }
5546                 \pgfusepath { fill }
5547             \end { pgfscope }
5548         }
5549     }
5550 \endpgfpicture
5551 }

```

The following command will extract the potential key `opacity` in its optional argument (between square brackets) and (of course) then apply the command `\color`.

```

5552 \cs_new_protected:Npn \@@_color_opacity
5553 {
5554   \peek_meaning:NTF [
5555     { \@@_color_opacity:w }
5556     { \@@_color_opacity:w [ ] }
5557   }

```

The command `\@@_color_opacity:w` takes in as argument only the optional argument. One may consider that the second argument (the actual definition of the color) is provided by curryfication.

```

5558 \cs_new_protected:Npn \@@_color_opacity:w [ #1 ]
5559 {
5560   \tl_clear:N \l_tmpa_tl
5561   \keys_set_known:nn { 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.

```

5562   \tl_if_empty:NF \l_tmpa_tl { \exp_args:No \pgfsetfillopacity \l_tmpa_tl }
5563   \tl_if_empty:NTF \l_tmpb_tl
5564     { \@declaredcolor }
5565     { \use:e { \exp_not:N \@undeclaredcolor [ \l_tmpb_tl ] } }
5566 }

```

The following set of keys is used by the command `\@@_color_opacity:wn`.

```

5567 \keys_define:nn { nicematrix / color-opacity }
5568 {
5569   opacity .tl_set:N      = \l_tmpa_tl ,
5570   opacity .value_required:n = true
5571 }

```

```

5572 \cs_new_protected:Npn \@@_cartesian_color:nn #1 #2
5573 {
5574   \cs_set_nopar:Npn \l_@@_rows_tl { #1 }
5575   \cs_set_nopar:Npn \l_@@_cols_tl { #2 }
5576   \@@_cartesian_path:
5577 }

```

Here is an example : `\@@_rowcolor {red!15} {1,3,5-7,10-}`

```

5578 \NewDocumentCommand \@@_rowcolor { 0 { } m m }
5579 {
5580   \tl_if_blank:nF { #2 }
5581   {
5582     \@@_add_to_colors_seq:en
5583     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5584     { \@@_cartesian_color:nn { #3 } { - } }
5585   }
5586 }

```

Here an example : `\@@_columncolor:nn {red!15} {1,3,5-7,10-}`

```

5587 \NewDocumentCommand \@@_columncolor { 0 { } m m }
5588 {
5589   \tl_if_blank:nF { #2 }
5590   {
5591     \@@_add_to_colors_seq:en
5592     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5593     { \@@_cartesian_color:nn { - } { #3 } }
5594   }
5595 }

```

Here is an example : `\@@_rectanglecolor{red!15}{2-3}{5-6}`

```

5596 \NewDocumentCommand \@@_rectanglecolor { 0 { } m 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     { \@@_rectanglecolor:nnn { #3 } { #4 } { \c_zero_dim } }
5603   }
5604 }

```

The last argument is the radius of the corners of the rectangle.

```

5605 \NewDocumentCommand \@@_roundedrectanglecolor { 0 { } m m 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     { \@@_rectanglecolor:nnn { #3 } { #4 } { #5 } }
5612   }
5613 }

```

The last argument is the radius of the corners of the rectangle.

```

5614 \cs_new_protected:Npn \@@_rectanglecolor:nnn #1 #2 #3
5615 {
5616   \@@_cut_on_hyphen:w #1 \q_stop
5617   \tl_clear_new:N \l_@@_tmpc_tl
5618   \tl_clear_new:N \l_@@_tmpd_tl
5619   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
5620   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
5621   \@@_cut_on_hyphen:w #2 \q_stop
5622   \tl_set:Nx \l_@@_rows_tl { \l_@@_tmpc_tl - \l_tmpa_tl }
5623   \tl_set:Nx \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`.

```

5624   \@@_cartesian_path:n { #3 }
5625 }

```

Here is an example : `\@@_cellcolor[rgb]{0.5,0.5,0}{2-3,3-4,4-5,5-6}`

```

5626 \NewDocumentCommand \@@_cellcolor { 0 { } m m }
5627 {
5628   \clist_map_inline:nn { #3 }
5629   { \@@_rectanglecolor [ #1 ] { #2 } { ##1 } { ##1 } }
5630 }

```

```

5631 \NewDocumentCommand \@@_chessboardcolors { 0 { } m m }
5632 {
5633   \int_step_inline:nn \c@iRow
5634   {
5635     \int_step_inline:nn \c@jCol
5636     {
5637       \int_if_even:nTF { ####1 + ##1 }
5638       { \@@_cellcolor [ #1 ] { #2 } }
5639       { \@@_cellcolor [ #1 ] { #3 } }
5640       { ##1 - ####1 }
5641     }
5642   }
5643 }

```

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”.

```

5644 \NewDocumentCommand \@@_arraycolor { 0 { } m }
5645 {
5646   \@@_rectanglecolor [ #1 ] { #2 }
5647   { 1 - 1 }
5648   { \int_use:N \c@iRow - \int_use:N \c@jCol }
5649 }

5650 \keys_define:nn { NiceMatrix / rowcolors }
5651 {
5652   respect-blocks .bool_set:N = \l_@@_respect_blocks_bool ,
5653   respect-blocks .default:n = true ,
5654   cols .tl_set:N = \l_@@_cols_tl ,
5655   restart .bool_set:N = \l_@@_rowcolors_restart_bool ,
5656   restart .default:n = true ,
5657   unknown .code:n = \@@_error:n { Unknown~key~for~rowcolors }
5658 }

```

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*.

```

5659 \NewDocumentCommand \@@_rowlistcolors { 0 { } m m 0 { } }
5660 {

```

The group is for the options. `\l_@@_colors_seq` will be the list of colors.

```

5661   \group_begin:
5662   \seq_clear_new:N \l_@@_colors_seq
5663   \seq_set_split:Nnn \l_@@_colors_seq { , } { #3 }
5664   \tl_clear_new:N \l_@@_cols_tl
5665   \cs_set_nopar:Npn \l_@@_cols_tl { - }
5666   \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).

```

5667   \int_zero_new:N \l_@@_color_int
5668   \int_set_eq:NN \l_@@_color_int \c_one_int
5669   \bool_if:NT \l_@@_respect_blocks_bool
5670   {

```

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 a the sequence `\l_tmpa_seq`).

```

5671       \seq_set_eq:NN \l_tmpb_seq \g_@@_pos_of_blocks_seq
5672       \seq_set_filter:Nnn \l_tmpa_seq \l_tmpb_seq
5673       { \@@_not_in_exterior_p:nnnnn ##1 }
5674   }
5675   \pgfpicture
5676   \pgf@relevantforpicturesizefalse

```

#2 is the list of intervals of rows.

```

5677   \clist_map_inline:nn { #2 }
5678   {
5679     \cs_set_nopar:Npn \l_tmpa_tl { ##1 }
5680     \tl_if_in:NnTF \l_tmpa_tl { - }
5681     { \@@_cut_on_hyphen:w ##1 \q_stop }
5682     { \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.

```

5683 \int_set:Nn \l_tmpa_int \l_tmpa_tl
5684 \int_set:Nn \l_@@_color_int
5685 { \bool_if:NTF \l_@@_rowcolors_restart_bool 1 \l_tmpa_tl }
5686 \int_zero_new:N \l_@@_tmpc_int
5687 \int_set:Nn \l_@@_tmpc_int \l_tmpb_tl
5688 \int_do_until:nNnn \l_tmpa_int > \l_@@_tmpc_int
5689 {

```

We will compute in `\l_tmpb_int` the last row of the “block”.

```

5690 \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).

```

5691 \bool_if:NT \l_@@_respect_blocks_bool
5692 {
5693   \seq_set_filter:NNn \l_tmpb_seq \l_tmpa_seq
5694   { \@@_intersect_our_row_p:nnnnn #####1 }
5695   \seq_map_inline:Nn \l_tmpb_seq { \@@_rowcolors_i:nnnnn #####1 }

```

Now, the last row of the block is computed in `\l_tmpb_int`.

```

5696 }
5697 \tl_set:No \l_@@_rows_tl
5698 { \int_use:N \l_tmpa_int - \int_use:N \l_tmpb_int }

```

`\l_@@_tmpc_tl` will be the color that we will use.

```

5699 \tl_clear_new:N \l_@@_color_tl
5700 \tl_set:Nx \l_@@_color_tl
5701 {
5702   \@@_color_index:n
5703   {
5704     \int_mod:nn
5705     { \l_@@_color_int - 1 }
5706     { \seq_count:N \l_@@_colors_seq }
5707     + 1
5708   }
5709 }
5710 \tl_if_empty:NF \l_@@_color_tl
5711 {
5712   \@@_add_to_colors_seq:ee
5713   { \tl_if_blank:nF { #1 } { [ #1 ] } { \l_@@_color_tl } }
5714   { \@@_cartesian_color:nn { \l_@@_rows_tl } { \l_@@_cols_tl } }
5715 }
5716 \int_incr:N \l_@@_color_int
5717 \int_set:Nn \l_tmpa_int { \l_tmpb_int + 1 }
5718 }
5719 }
5720 \endpgfpicture
5721 \group_end:
5722 }

```

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.

```

5723 \cs_new:Npn \@@_color_index:n #1
5724 {
5725   \str_if_eq:eeTF { \seq_item:Nn \l_@@_colors_seq { #1 } } { = }
5726   { \@@_color_index:n { #1 - 1 } }
5727   { \seq_item:Nn \l_@@_colors_seq { #1 } }
5728 }

```

The command `\rowcolors` (available in the `\CodeBefore`) is a specialisation of the more general command `\rowlistcolors`. The last argument, which is an optional argument between square brackets is provided by currying.

```

5729 \NewDocumentCommand \@@_rowcolors { 0 { } m m m }
5730 { \@@_rowlistcolors [ #1 ] { #2 } { { #3 } , { #4 } } }

```

The braces around #3 and #4 are mandatory.

```

5731 \cs_new_protected:Npn \@@_rowcolors_i:nnnnn #1 #2 #3 #4 #5
5732 {
5733   \int_compare:nNtT { #3 } > \l_tmpb_int
5734   { \int_set:Nn \l_tmpb_int { #3 } }
5735 }

5736 \prg_new_conditional:Nnn \@@_not_in_exterior:nnnnn p
5737 {
5738   \int_if_zero:nTF { #4 }
5739   \prg_return_false:
5740   {
5741     \int_compare:nNtTF { #2 } > \c@jCol
5742     \prg_return_false:
5743     \prg_return_true:
5744   }
5745 }

```

The following command return true when the block intersects the row \l_tmpa_int.

```

5746 \prg_new_conditional:Nnn \@@_intersect_our_row:nnnnn p
5747 {
5748   \int_compare:nNtTF { #1 } > \l_tmpa_int
5749   \prg_return_false:
5750   {
5751     \int_compare:nNtTF \l_tmpa_int > { #3 }
5752     \prg_return_false:
5753     \prg_return_true:
5754   }
5755 }

```

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).

```

5756 \cs_new_protected:Npn \@@_cartesian_path_normal:n #1
5757 {
5758   \dim_compare:nNtTF { #1 } = \c_zero_dim
5759   {
5760     \bool_if:NTF
5761     \l_@@_nocolor_used_bool
5762     \@@_cartesian_path_normal_ii:
5763     {
5764       \seq_if_empty:NTF \l_@@_corners_cells_seq
5765       { \@@_cartesian_path_normal_i:n { #1 } }
5766       \@@_cartesian_path_normal_ii:
5767     }
5768   }
5769   { \@@_cartesian_path_normal_i:n { #1 } }
5770 }

```

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.

```

5771 \cs_new_protected:Npn \@@_cartesian_path_normal_i:n #1
5772 {
5773   \pgfsetcornersarced { \pgfpoint { #1 } { #1 } }

```


We begin the loop over the columns.

```

5774 \clist_map_inline:Nn \l_@@_cols_tl
5775 {
5776   \cs_set_nopar:Npn \l_tmpa_tl { ##1 }
5777   \tl_if_in:NnTF \l_tmpa_tl { - }
5778     { \@@_cut_on_hyphen:w ##1 \q_stop }
5779     { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
5780   \tl_if_empty:NNTF \l_tmpa_tl
5781     { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5782     {
5783       \tl_if_eq:NNT \l_tmpa_tl \c_@@_star_tl
5784       { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5785     }
5786   \tl_if_empty:NNTF \l_tmpb_tl
5787     { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
5788     {
5789       \tl_if_eq:NNT \l_tmpb_tl \c_@@_star_tl
5790       { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
5791     }
5792   \int_compare:nNnT \l_tmpb_tl > \g_@@_col_total_int
5793     { \tl_set:No \l_tmpb_tl { \int_use:N \g_@@_col_total_int } }

```

`\l_@@_tmpc_tl` will contain the number of column.

```

5794   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
5795   \@@_qpoint:n { col - \l_tmpa_tl }
5796   \int_compare:nNnTF \l_@@_first_col_int = \l_tmpa_tl
5797     { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
5798     { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
5799   \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
5800   \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }

```

We begin the loop over the rows.

```

5801 \clist_map_inline:Nn \l_@@_rows_tl
5802 {
5803   \cs_set_nopar:Npn \l_tmpa_tl { #####1 }
5804   \tl_if_in:NnTF \l_tmpa_tl { - }
5805     { \@@_cut_on_hyphen:w #####1 \q_stop }
5806     { \@@_cut_on_hyphen:w #####1 - #####1 \q_stop }
5807   \tl_if_empty:NNTF \l_tmpa_tl
5808     { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5809     {
5810       \tl_if_eq:NNT \l_tmpa_tl \c_@@_star_tl
5811       { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5812     }
5813   \tl_if_empty:NNTF \l_tmpb_tl
5814     { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }
5815     {
5816       \tl_if_eq:NNT \l_tmpb_tl \c_@@_star_tl
5817       { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }
5818     }
5819   \int_compare:nNnT \l_tmpb_tl > \g_@@_row_total_int
5820     { \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`.

```

5821 \cs_if_exist:cF
5822 { @@ _ \l_tmpa_tl _ \l_@@_tmpc_tl _ nocolor }
5823 {
5824   \@@_qpoint:n { row - \int_eval:n { \l_tmpb_tl + 1 } }
5825   \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
5826   \@@_qpoint:n { row - \l_tmpa_tl }
5827   \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
5828   \pgfpathrectanglecorners
5829     { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
5830     { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5831 }

```

```

5832     }
5833   }
5834 }

```

Now, the case where the cells will be colored cell by cell (it's mandatory for example if the key `corners` is used).

```

5835 \cs_new_protected:Npn \@@_cartesian_path_normal_ii:
5836 {
5837   \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
5838   \@@_expand_clist:NN \l_@@_rows_tl \c@iRow

```

We begin the loop over the columns.

```

5839   \clist_map_inline:Nn \l_@@_cols_tl
5840   {
5841     \@@_qpoint:n { col - ##1 }
5842     \int_compare:nNnTF \l_@@_first_col_int = { ##1 }
5843       { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
5844       { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
5845     \@@_qpoint:n { col - \int_eval:n { ##1 + 1 } }
5846     \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }

```

We begin the loop over the rows.

```

5847   \clist_map_inline:Nn \l_@@_rows_tl
5848   {
5849     \seq_if_in:NnF \l_@@_corners_cells_seq
5850     { #####1 - ##1 }
5851     {
5852       \@@_qpoint:n { row - \int_eval:n { #####1 + 1 } }
5853       \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
5854       \@@_qpoint:n { row - #####1 }
5855       \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
5856       \cs_if_exist:cF { @@ - #####1 - ##1 - nocolor }
5857       {
5858         \pgfpathrectanglecorners
5859         { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
5860         { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5861       }
5862     }
5863   }
5864 }
5865 }

```

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`).

```

5866 \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

```

5867 \cs_new_protected:Npn \@@_cartesian_path_nocolor:n #1
5868 {
5869   \bool_set_true:N \l_@@_nocolor_used_bool
5870   \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
5871   \@@_expand_clist:NN \l_@@_rows_tl \c@iRow

```

We begin the loop over the columns.

```

5872   \clist_map_inline:Nn \l_@@_rows_tl
5873   {
5874     \clist_map_inline:Nn \l_@@_cols_tl
5875     { \cs_set:cpn { @@ - ##1 - #####1 - nocolor } { } }
5876   }
5877 }

```

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.

```

5878 \cs_new_protected:Npn \@@_expand_clist:NN #1 #2
5879 {
5880   \clist_set_eq:NN \l_tmpa_clist #1
5881   \clist_clear:N #1
5882   \clist_map_inline:Nn \l_tmpa_clist
5883   {
5884     \cs_set_nopar:Npn \l_tmpa_tl { ##1 }
5885     \tl_if_in:NnTF \l_tmpa_tl { - }
5886       { \@@_cut_on_hyphen:w ##1 \q_stop }
5887       { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
5888     \bool_lazy_or:nnT
5889       { \tl_if_blank_p:o \l_tmpa_tl }
5890       { \str_if_eq_p:on \l_tmpa_tl { * } }
5891       { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5892     \bool_lazy_or:nnT
5893       { \tl_if_blank_p:o \l_tmpb_tl }
5894       { \str_if_eq_p:on \l_tmpb_tl { * } }
5895       { \tl_set:N \l_tmpb_tl { \int_use:N #2 } }
5896     \int_compare:nNnT \l_tmpb_tl > #2
5897       { \tl_set:N \l_tmpb_tl { \int_use:N #2 } }
5898     \int_step_inline:nnn \l_tmpa_tl \l_tmpb_tl
5899       { \clist_put_right:Nn #1 { ####1 } }
5900   }
5901 }

```

When the user uses the key `color-inside`, the following command will be linked to `\cellcolor` in the tabular.

```

5902 \NewDocumentCommand \@@_cellcolor_tabular { 0 { } m }
5903 {
5904   \@@_test_color_inside:
5905   \tl_gput_right:Nx \g_@@_pre_code_before_tl
5906   {

```

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`).

```

5907     \@@_cellcolor [ #1 ] { \exp_not:n { #2 } }
5908     { \int_use:N \c@iRow - \int_use:N \c@jCol }
5909   }
5910   \ignorespaces
5911 }

```

When the user uses the key `color-inside`, the following command will be linked to `\rowcolor` in the tabular.

```

5912 \NewDocumentCommand \@@_rowcolor_tabular { 0 { } m }
5913 {
5914   \@@_test_color_inside:
5915   \tl_gput_right:Nx \g_@@_pre_code_before_tl
5916   {
5917     \@@_rectanglecolor [ #1 ] { \exp_not:n { #2 } }
5918     { \int_use:N \c@iRow - \int_use:N \c@jCol }
5919     { \int_use:N \c@iRow - \exp_not:n { \int_use:N \c@jCol } }
5920   }
5921   \ignorespaces
5922 }

```

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).

```

5923 \NewDocumentCommand { \@@_rowcolors_tabular } { 0 { } m m }
5924 { \@@_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.

```

5925 \NewDocumentCommand { \@@_rowlistcolors_tabular } { 0 { } m 0 { } }
5926 {
5927   \@@_test_color_inside:
5928   \peek_remove_spaces:n
5929   { \@@_rowlistcolors_tabular:nnn { #1 } { #2 } { #3 } }
5930 }

5931 \cs_new_protected:Npn \@@_rowlistcolors_tabular:nnn #1 #2 #3
5932 {

```

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`.

```

5933   \seq_gclear:N \g_tmpa_seq
5934   \seq_map_inline:Nn \g_@@_rowlistcolors_seq
5935   { \@@_rowlistcolors_tabular_i:nnnn ##1 }
5936   \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`.

```

5937   \seq_gput_right:Nx \g_@@_rowlistcolors_seq
5938   {
5939     { \int_use:N \c@iRow }
5940     { \exp_not:n { #1 } }
5941     { \exp_not:n { #2 } }
5942     { restart , cols = \int_use:N \c@jCol - , \exp_not:n { #3 } }
5943   }
5944 }

```

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`).

```

5945 \cs_new_protected:Npn \@@_rowlistcolors_tabular_i:nnnn #1 #2 #3 #4
5946 {
5947   \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).

```

5948   { \seq_gput_right:Nn \g_tmpa_seq { { #1 } { #2 } { #3 } { #4 } } }
5949   {
5950     \tl_gput_right:Nx \g_@@_pre_code_before_tl
5951     {
5952       \@@_rowlistcolors
5953       [ \exp_not:n { #2 } ]
5954       { #1 - \int_eval:n { \c@iRow - 1 } }
5955       { \exp_not:n { #3 } }
5956       [ \exp_not:n { #4 } ]
5957     }
5958   }
5959 }

```

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.

```

5960 \cs_new_protected:Npn \@@_clear_rowlistcolors_seq:
5961 {
5962   \seq_map_inline:Nn \g_@@_rowlistcolors_seq
5963     { \@@_rowlistcolors_tabular_ii:nnnn ##1 }
5964   \seq_gclear:N \g_@@_rowlistcolors_seq
5965 }

5966 \cs_new_protected:Npn \@@_rowlistcolors_tabular_ii:nnnn #1 #2 #3 #4
5967 {
5968   \tl_gput_right:Nn \g_@@_pre_code_before_tl
5969     { \@@_rowlistcolors [ #2 ] { #1 } { #3 } [ #4 ] }
5970 }

```

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.

```

5971 \NewDocumentCommand \@@_columncolor_preamble { 0 { } m }
5972 {

```

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).

```

5973   \int_compare:nNnT \c@jCol > \g_@@_col_total_int
5974   {

```

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).

```

5975     \tl_gput_left:Nx \g_@@_pre_code_before_tl
5976     {
5977       \exp_not:N \columncolor [ #1 ]
5978       { \exp_not:n { #2 } } { \int_use:N \c@jCol }
5979     }
5980   }
5981 }

5982 \hook_gput_code:nnn { begindocument } { . }
5983 {
5984   \IfPackageLoadedTF { colortbl }
5985   {
5986     \cs_set_eq:NN \@@_old_cellcolor \cellcolor
5987     \cs_set_eq:NN \@@_old_rowcolor \rowcolor
5988     \cs_new_protected:Npn \@@_revert_colortbl:
5989     {
5990       \hook_gput_code:nnn { env / tabular / begin } { nicematrix }
5991       {
5992         \cs_set_eq:NN \cellcolor \@@_old_cellcolor
5993         \cs_set_eq:NN \rowcolor \@@_old_rowcolor
5994       }
5995     }
5996   }
5997   { \cs_new_protected:Npn \@@_revert_colortbl: { } }
5998 }

```

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`.

```
5999 \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`.

```
6000 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix:n #1
6001 {
6002   \int_if_zero:nTF \l_@@_first_col_int
6003     { \@@_OnlyMainNiceMatrix_i:n { #1 } }
6004     {
6005       \int_if_zero:nTF \c@jCol
6006       {
6007         \int_compare:nNnF \c@iRow = { -1 }
6008         { \int_compare:nNnF \c@iRow = { \l_@@_last_row_int - 1 } { #1 } }
6009       }
6010       { \@@_OnlyMainNiceMatrix_i:n { #1 } }
6011     }
6012 }
```

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.

```
6013 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix_i:n #1
6014 {
6015   \int_if_zero:nF \c@iRow
6016   {
6017     \int_compare:nNnF \c@iRow = \l_@@_last_row_int
6018     {
6019       \int_compare:nNnT \c@jCol > \c_zero_int
6020       { \bool_if:NF \l_@@_in_last_col_bool { #1 } }
6021     }
6022   }
6023 }
```

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.

```
6024 \keys_define:nn { NiceMatrix / Rules }
6025 {
6026   position .int_set:N = \l_@@_position_int ,
6027   position .value_required:n = true ,
6028   start .int_set:N = \l_@@_start_int ,
```

```

6029     end .code:n =
6030         \bool_lazy_or:nnTF
6031         { \tl_if_empty_p:n { #1 } }
6032         { \str_if_eq_p:nn { #1 } { last } }
6033         { \int_set_eq:NN \l_@@_end_int \c@jCol }
6034         { \int_set:Nn \l_@@_end_int { #1 } }
6035     }

```

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.

```

6036 \keys_define:nn { NiceMatrix / RulesBis }
6037 {
6038     multiplicity .int_set:N = \l_@@_multiplicity_int ,
6039     multiplicity .initial:n = 1 ,
6040     dotted .bool_set:N = \l_@@_dotted_bool ,
6041     dotted .initial:n = false ,
6042     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]`.

```

6043     color .code:n =
6044         \@@_set_CT@arc@:n { #1 }
6045         \tl_set:Nn \l_@@_rule_color_tl { #1 } ,
6046     color .value_required:n = true ,
6047     sep-color .code:n = \@@_set_CT@drsc@:n { #1 } ,
6048     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.

```

6049     tikz .code:n =
6050         \IfPackageLoadedTF { tikz }
6051         { \clist_put_right:Nn \l_@@_tikz_rule_tl { #1 } }
6052         { \@@_error:n { tikz-without-tikz } } ,
6053     tikz .value_required:n = true ,
6054     total-width .dim_set:N = \l_@@_rule_width_dim ,
6055     total-width .value_required:n = true ,
6056     width .meta:n = { total-width = #1 } ,
6057     unknown .code:n = \@@_error:n { Unknow-key-for-RulesBis }
6058 }

```

The vertical rules

The following command will be executed in the internal `\CodeAfter`. The argument `#1` is a list of `key=value` pairs.

```

6059 \cs_new_protected:Npn \@@_vline:n #1
6060 {

```

The group is for the options.

```

6061     \group_begin:
6062     \int_set_eq:NN \l_@@_end_int \c@iRow
6063     \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).

```

6064     \int_compare:nNnT \l_@@_position_int < { \c@jCol + 2 }
6065     \@@_vline_i:
6066     \group_end:
6067 }

```

```

6068 \cs_new_protected:Npn \@@_vline_i:
6069 {

```

\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.

```

6070 \tl_set:N \l_tmpb_tl { \int_use:N \l_@@_position_int }
6071 \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
6072 \l_tmpa_tl
6073 {

```

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.

```

6074 \bool_gset_true:N \g_tmpa_bool
6075 \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6076 { \@@_test_vline_in_block:nnnnn ##1 }
6077 \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
6078 { \@@_test_vline_in_block:nnnnn ##1 }
6079 \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
6080 { \@@_test_vline_in_stroken_block:nnnn ##1 }
6081 \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_v:
6082 \bool_if:NTF \g_tmpa_bool
6083 {
6084 \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.

```

6085 { \int_set:Nn \l_@@_local_start_int \l_tmpa_tl }
6086 }
6087 {
6088 \int_compare:nNnT \l_@@_local_start_int > \c_zero_int
6089 {
6090 \int_set:Nn \l_@@_local_end_int { \l_tmpa_tl - 1 }
6091 \@@_vline_ii:
6092 \int_zero:N \l_@@_local_start_int
6093 }
6094 }
6095 }
6096 \int_compare:nNnT \l_@@_local_start_int > \c_zero_int
6097 {
6098 \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
6099 \@@_vline_ii:
6100 }
6101 }

```

```

6102 \cs_new_protected:Npn \@@_test_in_corner_v:
6103 {
6104 \int_compare:nNnTF \l_tmpb_tl = { \int_eval:n { \c@jCol + 1 } }
6105 {
6106 \seq_if_in:NxT
6107 \l_@@_corners_cells_seq
6108 { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
6109 { \bool_set_false:N \g_tmpa_bool }
6110 }
6111 {
6112 \seq_if_in:NxT
6113 \l_@@_corners_cells_seq
6114 { \l_tmpa_tl - \l_tmpb_tl }
6115 {
6116 \int_compare:nNnTF \l_tmpb_tl = \c_one_int
6117 { \bool_set_false:N \g_tmpa_bool }
6118 {

```



```

6119         \seq_if_in:NxT
6120         \l_@@_corners_cells_seq
6121         { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
6122         { \bool_set_false:N \g_tmpa_bool }
6123     }
6124 }
6125 }
6126 }

6127 \cs_new_protected:Npn \@@_vline_ii:
6128 {
6129     \tl_clear:N \l_@@_tikz_rule_tl
6130     \keys_set:nV { NiceMatrix / RulesBis } \l_@@_other_keys_tl
6131     \bool_if:NTF \l_@@_dotted_bool
6132     \@@_vline_iv:
6133     {
6134         \tl_if_empty:NTF \l_@@_tikz_rule_tl
6135         \@@_vline_iii:
6136         \@@_vline_v:
6137     }
6138 }

```

First the case of a standard rule: the user has not used the key `dotted` nor the key `tikz`.

```

6139 \cs_new_protected:Npn \@@_vline_iii:
6140 {
6141     \pgfpicture
6142     \pgfrememberpicturepositiononpagetrue
6143     \pgf@relevantforpicturesizefalse
6144     \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6145     \dim_set_eq:NN \l_tmpa_dim \pgf@y
6146     \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6147     \dim_set:Nn \l_tmpb_dim
6148     {
6149         \pgf@x
6150         - 0.5 \l_@@_rule_width_dim
6151         +
6152         ( \arrayrulewidth * \l_@@_multiplicity_int
6153           + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6154     }
6155     \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6156     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
6157     \bool_lazy_all:nT
6158     {
6159         { \int_compare_p:nNn \l_@@_multiplicity_int > \c_one_int }
6160         { \cs_if_exist_p:N \CT@drsc@ }
6161         { ! \tl_if_blank_p:o \CT@drsc@ }
6162     }
6163     {
6164         \group_begin:
6165         \CT@drsc@
6166         \dim_add:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }
6167         \dim_sub:Nn \l_@@_tmpc_dim { 0.5 \arrayrulewidth }
6168         \dim_set:Nn \l_@@_tmpd_dim
6169         {
6170             \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
6171             * ( \l_@@_multiplicity_int - 1 )
6172         }
6173         \pgfpathrectanglecorners
6174         { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6175         { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
6176         \pgfusepath { fill }
6177         \group_end:

```

```

6178     }
6179     \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6180     \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
6181     \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6182     {
6183         \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
6184         \dim_sub:Nn \l_tmpb_dim \doublerulesep
6185         \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6186         \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
6187     }
6188     \CT@arc@
6189     \pgfsetlinewidth { 1.1 \arrayrulewidth }
6190     \pgfsetrectcap
6191     \pgfusepathqstroke
6192     \endpgfpicture
6193 }

```

The following code is for the case of a dotted rule (with our system of rounded dots).

```

6194 \cs_new_protected:Npn \@@_vline_iv:
6195 {
6196     \pgfpicture
6197     \pgfrememberpicturepositiononpagetrue
6198     \pgf@relevantforpicturesizefalse
6199     \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6200     \dim_set:Nn \l_@@_x_initial_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
6201     \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
6202     \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6203     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
6204     \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6205     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
6206     \CT@arc@
6207     \@@_draw_line:
6208     \endpgfpicture
6209 }

```

The following code is for the case when the user uses the key `tikz`.

```

6210 \cs_new_protected:Npn \@@_vline_v:
6211 {
6212     \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).

```

6213     \CT@arc@
6214     \tl_if_empty:NF \l_@@_rule_color_tl
6215     { \tl_put_right:Nx \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6216     \pgfrememberpicturepositiononpagetrue
6217     \pgf@relevantforpicturesizefalse
6218     \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6219     \dim_set_eq:NN \l_tmpa_dim \pgf@y
6220     \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6221     \dim_set:Nn \l_tmpb_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
6222     \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6223     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
6224     \exp_args:No \tikzset \l_@@_tikz_rule_tl
6225     \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6226     ( \l_tmpb_dim , \l_tmpa_dim ) --
6227     ( \l_tmpb_dim , \l_@@_tmpc_dim ) ;
6228     \end { tikzpicture }
6229 }

```

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).

```

6230 \cs_new_protected:Npn \@@_draw_vlines:
6231 {
6232   \int_step_inline:nnn
6233     { \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool 2 1 }
6234     {
6235       \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool
6236       \c@jCol
6237       { \int_eval:n { \c@jCol + 1 } }
6238     }
6239     {
6240       \tl_if_eq:NNF \l_@@_vlines_clist \c_@@_all_tl
6241       { \clist_if_in:NnT \l_@@_vlines_clist { ##1 } }
6242       { \@@_vline:n { position = ##1 , total-width = \arrayrulewidth } }
6243     }
6244 }

```

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}`.

```

6245 \cs_new_protected:Npn \@@_hline:n #1
6246 {

```

The group is for the options.

```

6247   \group_begin:
6248   \int_zero_new:N \l_@@_end_int
6249   \int_set_eq:NN \l_@@_end_int \c@jCol
6250   \keys_set_known:nnN { NiceMatrix / Rules } { #1 } \l_@@_other_keys_tl
6251   \@@_hline_i:
6252   \group_end:
6253 }
6254 \cs_new_protected:Npn \@@_hline_i:
6255 {
6256   \int_zero_new:N \l_@@_local_start_int
6257   \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`.

```

6258   \tl_set:No \l_tmpa_tl { \int_use:N \l_@@_position_int }
6259   \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
6260   \l_tmpb_tl
6261   {

```

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.

```

6262     \bool_gset_true:N \g_tmpa_bool
6263     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6264       { \@@_test_hline_in_block:nnnnn ##1 }
6265     \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
6266       { \@@_test_hline_in_block:nnnnn ##1 }
6267     \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
6268       { \@@_test_hline_in_stroken_block:nnnn ##1 }
6269     \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_h:
6270     \bool_if:NTF \g_tmpa_bool
6271     {
6272       \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.

```

6273         { \int_set:Nn \l_@@_local_start_int \l_tmpb_tl }
6274     }
6275     {
6276         \int_compare:nNnT \l_@@_local_start_int > \c_zero_int
6277         {
6278             \int_set:Nn \l_@@_local_end_int { \l_tmpb_tl - 1 }
6279             \@@_hline_ii:
6280             \int_zero:N \l_@@_local_start_int
6281         }
6282     }
6283 }
6284 \int_compare:nNnT \l_@@_local_start_int > \c_zero_int
6285 {
6286     \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
6287     \@@_hline_ii:
6288 }
6289 }

6290 \cs_new_protected:Npn \@@_test_in_corner_h:
6291 {
6292     \int_compare:nNnTF \l_tmpa_tl = { \int_eval:n { \c@iRow + 1 } }
6293     {
6294         \seq_if_in:NxT
6295         \l_@@_corners_cells_seq
6296         { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
6297         { \bool_set_false:N \g_tmpa_bool }
6298     }
6299     {
6300         \seq_if_in:NxT
6301         \l_@@_corners_cells_seq
6302         { \l_tmpa_tl - \l_tmpb_tl }
6303         {
6304             \int_compare:nNnTF \l_tmpa_tl = \c_one_int
6305             { \bool_set_false:N \g_tmpa_bool }
6306             {
6307                 \seq_if_in:NxT
6308                 \l_@@_corners_cells_seq
6309                 { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
6310                 { \bool_set_false:N \g_tmpa_bool }
6311             }
6312         }
6313     }
6314 }

6315 \cs_new_protected:Npn \@@_hline_ii:
6316 {
6317     \tl_clear:N \l_@@_tikz_rule_tl
6318     \keys_set:nV { NiceMatrix / RulesBis } \l_@@_other_keys_tl
6319     \bool_if:NTF \l_@@_dotted_bool
6320     \@@_hline_iv:
6321     {
6322         \tl_if_empty:NTF \l_@@_tikz_rule_tl
6323         \@@_hline_iii:
6324         \@@_hline_v:
6325     }
6326 }
```

First the case of a standard rule (without the keys `dotted` and `tikz`).

```

6327 \cs_new_protected:Npn \@@_hline_iii:
```

```

6328 {
6329   \pgfpicture
6330   \pgfrememberpicturepositiononpagetrue
6331   \pgf@relevantforpicturesizefalse
6332   \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6333   \dim_set_eq:NN \l_tmpa_dim \pgf@x
6334   \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6335   \dim_set:Nn \l_tmpb_dim
6336     {
6337       \pgf@y
6338       - 0.5 \l_@@_rule_width_dim
6339       +
6340       ( \arrayrulewidth * \l_@@_multiplicity_int
6341         + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6342     }
6343   \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6344   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
6345   \bool_lazy_all:nT
6346     {
6347       { \int_compare_p:nNn \l_@@_multiplicity_int > \c_one_int }
6348       { \cs_if_exist_p:N \CT@drsc@ }
6349       { ! \tl_if_blank_p:o \CT@drsc@ }
6350     }
6351     {
6352       \group_begin:
6353       \CT@drsc@
6354       \dim_set:Nn \l_@@_tmpd_dim
6355         {
6356           \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
6357           * ( \l_@@_multiplicity_int - 1 )
6358         }
6359       \pgfpathrectanglecorners
6360         { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6361         { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
6362       \pgfusepathqfill
6363       \group_end:
6364     }
6365   \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6366   \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
6367   \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6368     {
6369       \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
6370       \dim_sub:Nn \l_tmpb_dim \doublerulesep
6371       \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6372       \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
6373     }
6374   \CT@arc@
6375   \pgfsetlinewidth { 1.1 \arrayrulewidth }
6376   \pgfsetrectcap
6377   \pgfusepathqstroke
6378   \endpgfpicture
6379 }

```

The following code is for the case of a dotted rule (with our system of rounded dots). The aim is that, by standard the dotted line fits between square brackets (`\hline` doesn't).

```

\begin{bNiceMatrix}
1 & 2 & 3 & 4 \\
\hline
1 & 2 & 3 & 4 \\
\hdottedline
1 & 2 & 3 & 4
\end{bNiceMatrix}

```

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ \hdottedline 1 & 2 & 3 & 4 \end{bmatrix}$$

But, if the user uses `margin`, the dotted line extends to have the same width as a `\hline`.

```

\begin{bNiceMatrix}[margin]
1 & 2 & 3 & 4 \\
\hline
1 & 2 & 3 & 4 \\
\hdottedline
1 & 2 & 3 & 4
\end{bNiceMatrix}

```

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ \cdots & \cdots & \cdots & \cdots \\ 1 & 2 & 3 & 4 \end{bmatrix}$$

```

6380 \cs_new_protected:Npn \@@_hline_iv:
6381 {
6382   \pgfpicture
6383   \pgfrememberpicturepositiononpagetrue
6384   \pgf@relevantforpicturesizefalse
6385   \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6386   \dim_set:Nn \l_@@_y_initial_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
6387   \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
6388   \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6389   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
6390   \int_compare:nNnT \l_@@_local_start_int = \c_one_int
6391   {
6392     \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
6393     \bool_if:NF \g_@@_delims_bool
6394     { \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.

```

6395   \tl_if_eq:NnF \g_@@_left_delim_tl (
6396     { \dim_add:Nn \l_@@_x_initial_dim { 0.5 \l_@@_xdots_inter_dim } }
6397   )
6398   \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6399   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
6400   \int_compare:nNnT \l_@@_local_end_int = \c_jCol
6401   {
6402     \dim_add:Nn \l_@@_x_final_dim \l_@@_right_margin_dim
6403     \bool_if:NF \g_@@_delims_bool
6404     { \dim_add:Nn \l_@@_x_final_dim \arraycolsep }
6405     \tl_if_eq:NnF \g_@@_right_delim_tl )
6406     { \dim_gsub:Nn \l_@@_x_final_dim { 0.5 \l_@@_xdots_inter_dim } }
6407   }
6408   \CT@arc@
6409   \@@_draw_line:
6410   \endpgfpicture
6411 }

```

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`).

```

6412 \cs_new_protected:Npn \@@_hline_v:
6413 {
6414   \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).

```

6415   \CT@arc@
6416   \tl_if_empty:NF \l_@@_rule_color_tl
6417   { \tl_put_right:Nx \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6418   \pgfrememberpicturepositiononpagetrue
6419   \pgf@relevantforpicturesizefalse
6420   \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6421   \dim_set_eq:NN \l_tmpa_dim \pgf@x
6422   \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6423   \dim_set:Nn \l_tmpb_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
6424   \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6425   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x

```

```

6426 \exp_args:No \tikzset \l_@@_tikz_rule_tl
6427 \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6428 ( \l_tmpa_dim , \l_tmpb_dim ) --
6429 ( \l_@@_tmpc_dim , \l_tmpb_dim ) ;
6430 \end { tikzpicture }
6431 }

```

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).

```

6432 \cs_new_protected:Npn \@@_draw_hlines:
6433 {
6434   \int_step_inline:nnn
6435   { \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool 2 1 }
6436   {
6437     \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool
6438     \c@iRow
6439     { \int_eval:n { \c@iRow + 1 } }
6440   }
6441   {
6442     \tl_if_eq:NNF \l_@@_hlines_clist \c_@@_all_tl
6443     { \clist_if_in:NnT \l_@@_hlines_clist { ##1 } }
6444     { \@@_hline:n { position = ##1 , total-width = \arrayrulewidth } }
6445   }
6446 }

```

The command `\@@_Hline:` will be linked to `\Hline` in the environments of `nicematrix`.

```

6447 \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.

```

6448 \cs_set:Npn \@@_Hline_i:n #1
6449 {
6450   \peek_remove_spaces:n
6451   {
6452     \peek_meaning:NTF \Hline
6453     { \@@_Hline_ii:nn { #1 + 1 } }
6454     { \@@_Hline_iii:n { #1 } }
6455   }
6456 }
6457 \cs_set:Npn \@@_Hline_ii:nn #1 #2 { \@@_Hline_i:n { #1 } }
6458 \cs_set:Npn \@@_Hline_iii:n #1
6459 { \@@_collect_options:n { \@@_Hline_iv:nn { #1 } } }
6460 \cs_set:Npn \@@_Hline_iv:nn #1 #2
6461 {
6462   \@@_compute_rule_width:n { multiplicity = #1 , #2 }
6463   \skip_vertical:N \l_@@_rule_width_dim
6464   \tl_gput_right:Nx \g_@@_pre_code_after_tl
6465   {
6466     \@@_hline:n
6467     {
6468       multiplicity = #1 ,
6469       position = \int_eval:n { \c@iRow + 1 } ,
6470       total-width = \dim_use:N \l_@@_rule_width_dim ,
6471       #2
6472     }
6473   }
6474   \egroup
6475 }

```

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`).

```

6476 \cs_new_protected:Npn \@@_custom_line:n #1
6477 {
6478   \str_clear_new:N \l_@@_command_str
6479   \str_clear_new:N \l_@@_ccommand_str
6480   \str_clear_new:N \l_@@_letter_str
6481   \tl_clear_new:N \l_@@_other_keys_tl
6482   \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.

```

6483   \bool_lazy_all:nTF
6484   {
6485     { \str_if_empty_p:N \l_@@_letter_str }
6486     { \str_if_empty_p:N \l_@@_command_str }
6487     { \str_if_empty_p:N \l_@@_ccommand_str }
6488   }
6489   { \@@_error:n { No~letter-and-no~command } }
6490   { \exp_args:No \@@_custom_line_i:n \l_@@_other_keys_tl }
6491 }

6492 \keys_define:nn { NiceMatrix / custom-line }
6493 {
6494   letter .str_set:N = \l_@@_letter_str ,
6495   letter .value_required:n = true ,
6496   command .str_set:N = \l_@@_command_str ,
6497   command .value_required:n = true ,
6498   ccommand .str_set:N = \l_@@_ccommand_str ,
6499   ccommand .value_required:n = true ,
6500 }

```

```

6501 \cs_new_protected:Npn \@@_custom_line_i:n #1
6502 {

```

The following flags will be raised when the keys `tikz`, `dotted` and `color` are used (in the `custom-line`).

```

6503   \bool_set_false:N \l_@@_tikz_rule_bool
6504   \bool_set_false:N \l_@@_dotted_rule_bool
6505   \bool_set_false:N \l_@@_color_bool

6506   \keys_set:nn { NiceMatrix / custom-line-bis } { #1 }
6507   \bool_if:NT \l_@@_tikz_rule_bool
6508   {
6509     \IfPackageLoadedTF { tikz }
6510     { }
6511     { \@@_error:n { tikz~in~custom-line~without~tikz } }
6512     \bool_if:NT \l_@@_color_bool
6513     { \@@_error:n { color~in~custom-line~with~tikz } }
6514   }
6515   \bool_if:NT \l_@@_dotted_rule_bool
6516   {
6517     \int_compare:nNnT \l_@@_multiplicity_int > \c_one_int
6518     { \@@_error:n { key~multiplicity~with~dotted } }
6519   }
6520   \str_if_empty:NF \l_@@_letter_str
6521   {

```



```

6522 \int_compare:nTF { \str_count:N \l_@@_letter_str != 1 }
6523 { \@@_error:n { Several~letters } }
6524 {
6525 \exp_args:NnV \tl_if_in:NnTF
6526 \c_@@_forbidden_letters_str \l_@@_letter_str
6527 { \@@_error:ne { Forbidden~letter } \l_@@_letter_str }
6528 {

```

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.

```

6529 \cs_set:cpn { @@ _ \l_@@_letter_str } ##1
6530 { \@@_v_custom_line:n { #1 } }
6531 }
6532 }
6533 }
6534 \str_if_empty:NF \l_@@_command_str { \@@_h_custom_line:n { #1 } }
6535 \str_if_empty:NF \l_@@_ccommand_str { \@@_c_custom_line:n { #1 } }
6536 }
6537 \tl_const:Nn \c_@@_forbidden_letters_tl { lcrpmbVX|()[]!@<> }
6538 \str_const:Nn \c_@@_forbidden_letters_str { lcrpmbVX|()[]!@<> }

```

The previous command `\@@_custom_line_i:n` uses the following set of keys. However, the whole definition of the customized lines (as provided by the final user as argument of `custom-line`) will also be used further with other sets of keys (for instance `{NiceMatrix/Rules}`). That's why the following set of keys has some keys which are no-op.

```

6539 \keys_define:nn { NiceMatrix / custom-line-bis }
6540 {
6541 multiplicity .int_set:N = \l_@@_multiplicity_int ,
6542 multiplicity .initial:n = 1 ,
6543 multiplicity .value_required:n = true ,
6544 color .code:n = \bool_set_true:N \l_@@_color_bool ,
6545 color .value_required:n = true ,
6546 tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6547 tikz .value_required:n = true ,
6548 dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6549 dotted .value_forbidden:n = true ,
6550 total-width .code:n = { } ,
6551 total-width .value_required:n = true ,
6552 width .code:n = { } ,
6553 width .value_required:n = true ,
6554 sep-color .code:n = { } ,
6555 sep-color .value_required:n = true ,
6556 unknown .code:n = \@@_error:n { Unknown~key~for~custom-line }
6557 }

```

The following keys will indicate whether the keys `dotted`, `tikz` and `color` are used in the use of a `custom-line`.

```

6558 \bool_new:N \l_@@_dotted_rule_bool
6559 \bool_new:N \l_@@_tikz_rule_bool
6560 \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`.

```

6561 \keys_define:nn { NiceMatrix / custom-line-width }
6562 {
6563 multiplicity .int_set:N = \l_@@_multiplicity_int ,
6564 multiplicity .initial:n = 1 ,
6565 multiplicity .value_required:n = true ,
6566 tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6567 total-width .code:n = \dim_set:Nn \l_@@_rule_width_dim { #1 }
6568 \bool_set_true:N \l_@@_total_width_bool ,

```

```

6569     total-width .value_required:n = true ,
6570     width .meta:n = { total-width = #1 } ,
6571     dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6572 }

```

The following command will create the command that the final user will use in its array to draw an horizontal rule (hence the ‘h’ in the name) with the full width of the array. #1 is the whole set of keys to pass to the command \@@_hline:n (which is in the internal \CodeAfter).

```

6573 \cs_new_protected:Npn \@@_h_custom_line:n #1
6574 {

```

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).

```

6575     \cs_set:cpn { nicematrix - \l_@@_command_str } { \Hline [ #1 ] }
6576     \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_command_str
6577 }

```

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).

```

6578 \cs_new_protected:Npn \@@_c_custom_line:n #1
6579 {

```

Here, we need an expandable command since it begins with an \noalign.

```

6580     \exp_args:Nc \NewExpandableDocumentCommand
6581     { nicematrix - \l_@@_ccommand_str }
6582     { 0 { } m }
6583     {
6584         \noalign
6585         {
6586             \@@_compute_rule_width:n { #1 , ##1 }
6587             \skip_vertical:n { \l_@@_rule_width_dim }
6588             \clist_map_inline:nn
6589             { ##2 }
6590             { \@@_c_custom_line_i:nn { #1 , ##1 } { #####1 } }
6591         }
6592     }
6593     \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_ccommand_str
6594 }

```

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*.

```

6595 \cs_new_protected:Npn \@@_c_custom_line_i:nn #1 #2
6596 {
6597     \str_if_in:nnTF { #2 } { - }
6598     { \@@_cut_on_hyphen:w #2 \q_stop }
6599     { \@@_cut_on_hyphen:w #2 - #2 \q_stop }
6600     \tl_gput_right:Nx \g_@@_pre_code_after_tl
6601     {
6602         \@@_hline:n
6603         {
6604             #1 ,
6605             start = \l_tmpa_tl ,
6606             end = \l_tmpb_tl ,
6607             position = \int_eval:n { \c@iRow + 1 } ,
6608             total-width = \dim_use:N \l_@@_rule_width_dim
6609         }
6610     }
6611 }

```

```

6612 \cs_new_protected:Npn \@@_compute_rule_width:n #1
6613 {
6614   \bool_set_false:N \l_@@_tikz_rule_bool
6615   \bool_set_false:N \l_@@_total_width_bool
6616   \bool_set_false:N \l_@@_dotted_rule_bool
6617   \keys_set_known:n { NiceMatrix / custom-line-width } { #1 }
6618   \bool_if:NF \l_@@_total_width_bool
6619   {
6620     \bool_if:NTF \l_@@_dotted_rule_bool
6621     { \dim_set:Nn \l_@@_rule_width_dim { 2 \l_@@_xdots_radius_dim } }
6622     {
6623       \bool_if:NF \l_@@_tikz_rule_bool
6624       {
6625         \dim_set:Nn \l_@@_rule_width_dim
6626         {
6627           \arrayrulewidth * \l_@@_multiplicity_int
6628           + \doublerulesep * ( \l_@@_multiplicity_int - 1 )
6629         }
6630       }
6631     }
6632   }
6633 }
6634 \cs_new_protected:Npn \@@_v_custom_line:n #1
6635 {
6636   \@@_compute_rule_width:n { #1 }

```

In the following line, the `\dim_use:N` is mandatory since we do an expansion.

```

6637   \tl_gput_right:Nx \g_@@_array_preamble_tl
6638   { \exp_not:N ! { \skip_horizontal:n { \dim_use:N \l_@@_rule_width_dim } } }
6639   \tl_gput_right:Nx \g_@@_pre_code_after_tl
6640   {
6641     \@@_vline:n
6642     {
6643       #1 ,
6644       position = \int_eval:n { \c@jCol + 1 } ,
6645       total-width = \dim_use:N \l_@@_rule_width_dim
6646     }
6647   }
6648   \@@_rec_preamble:n
6649 }
6650 \@@_custom_line:n
6651 { 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.

```

6652 \cs_new_protected:Npn \@@_test_hline_in_block:nnnnn #1 #2 #3 #4 #5
6653 {
6654   \int_compare:nNnT \l_tmpa_tl > { #1 }
6655   {
6656     \int_compare:nNnT \l_tmpa_tl < { #3 + 1 }
6657     {
6658       \int_compare:nNnT \l_tmpb_tl > { #2 - 1 }
6659       {
6660         \int_compare:nNnT \l_tmpb_tl < { #4 + 1 }
6661         { \bool_gset_false:N \g_tmpa_bool }
6662       }
6663     }
6664   }
6665 }

```

The same for vertical rules.

```

6666 \cs_new_protected:Npn \@@_test_vline_in_block:nnnnn #1 #2 #3 #4 #5
6667 {
6668   \int_compare:nNnT \l_tmpa_tl > { #1 - 1 }
6669   {
6670     \int_compare:nNnT \l_tmpa_tl < { #3 + 1 }
6671     {
6672       \int_compare:nNnT \l_tmpb_tl > { #2 }
6673       {
6674         \int_compare:nNnT \l_tmpb_tl < { #4 + 1 }
6675         { \bool_gset_false:N \g_tmpa_bool }
6676       }
6677     }
6678   }
6679 }

6680 \cs_new_protected:Npn \@@_test_hline_in_stroken_block:nnnn #1 #2 #3 #4
6681 {
6682   \int_compare:nNnT \l_tmpb_tl > { #2 - 1 }
6683   {
6684     \int_compare:nNnT \l_tmpb_tl < { #4 + 1 }
6685     {
6686       \int_compare:nNnTF \l_tmpa_tl = { #1 }
6687       { \bool_gset_false:N \g_tmpa_bool }
6688       {
6689         \int_compare:nNnT \l_tmpa_tl = { #3 + 1 }
6690         { \bool_gset_false:N \g_tmpa_bool }
6691       }
6692     }
6693   }
6694 }

6695 \cs_new_protected:Npn \@@_test_vline_in_stroken_block:nnnn #1 #2 #3 #4
6696 {
6697   \int_compare:nNnT \l_tmpa_tl > { #1 - 1 }
6698   {
6699     \int_compare:nNnT \l_tmpa_tl < { #3 + 1 }
6700     {
6701       \int_compare:nNnTF \l_tmpb_tl = { #2 }
6702       { \bool_gset_false:N \g_tmpa_bool }
6703       {
6704         \int_compare:nNnT \l_tmpb_tl = { #4 + 1 }
6705         { \bool_gset_false:N \g_tmpa_bool }
6706       }
6707     }
6708   }
6709 }

```

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.

```

6710 \cs_new_protected:Npn \@@_compute_corners:
6711 {

```

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.

```

6712   \seq_clear_new:N \l_@@_corners_cells_seq

```

```

6713 \clist_map_inline:Nn \l_@@_corners_clist
6714 {
6715   \str_case:nnF { ##1 }
6716   {
6717     { NW }
6718     { \@@_compute_a_corner:nnnnnn 1 1 1 1 \c@iRow \c@jCol }
6719     { NE }
6720     { \@@_compute_a_corner:nnnnnn 1 \c@jCol 1 { -1 } \c@iRow 1 }
6721     { SW }
6722     { \@@_compute_a_corner:nnnnnn \c@iRow 1 { -1 } 1 1 \c@jCol }
6723     { SE }
6724     { \@@_compute_a_corner:nnnnnn \c@iRow \c@jCol { -1 } { -1 } 1 1 }
6725   }
6726   { \@@_error:nn { bad~corner } { ##1 } }
6727 }

```

Even if the user has used the key `corners` the list of cells in the corners may be empty.

```

6728 \seq_if_empty:NF \l_@@_corners_cells_seq
6729 {

```

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.

```

6730   \tl_gput_right:Nx \g_@@_aux_tl
6731   {
6732     \seq_set_from_clist:Nn \exp_not:N \l_@@_corners_cells_seq
6733     { \seq_use:Nnnn \l_@@_corners_cells_seq , , , }
6734   }
6735 }
6736 }

```

“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.

```

6737 \cs_new_protected:Npn \@@_compute_a_corner:nnnnnn #1 #2 #3 #4 #5 #6
6738 {

```

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.

```

6739   \bool_set_false:N \l_tmpa_bool
6740   \int_zero_new:N \l_@@_last_empty_row_int
6741   \int_set:Nn \l_@@_last_empty_row_int { #1 }
6742   \int_step_inline:nnnn { #1 } { #3 } { #5 }
6743   {
6744     \@@_test_if_cell_in_a_block:nn { ##1 } { \int_eval:n { #2 } }
6745     \bool_lazy_or:nnTF
6746     {
6747       \cs_if_exist_p:c
6748       { pgf @ sh @ ns @ \@@_env: - ##1 - \int_eval:n { #2 } }
6749     }
6750     \l_tmpb_bool
6751     { \bool_set_true:N \l_tmpa_bool }

```

```

6752     {
6753         \bool_if:NF \l_tmpa_bool
6754         { \int_set:Nn \l_@@_last_empty_row_int { ##1 } }
6755     }
6756 }

```

Now, you determine the last empty cell in the row of number 1.

```

6757     \bool_set_false:N \l_tmpa_bool
6758     \int_zero_new:N \l_@@_last_empty_column_int
6759     \int_set:Nn \l_@@_last_empty_column_int { #2 }
6760     \int_step_inline:nnnn { #2 } { #4 } { #6 }
6761     {
6762         \@@_test_if_cell_in_a_block:nn { \int_eval:n { #1 } } { ##1 }
6763         \bool_lazy_or:nnTF
6764             \l_tmpb_bool
6765             {
6766                 \cs_if_exist_p:c
6767                     { pgf @ sh @ ns @ \@@_env: - \int_eval:n { #1 } - ##1 }
6768             }
6769         { \bool_set_true:N \l_tmpa_bool }
6770         {
6771             \bool_if:NF \l_tmpa_bool
6772             { \int_set:Nn \l_@@_last_empty_column_int { ##1 } }
6773         }
6774     }

```

Now, we loop over the rows.

```

6775     \int_step_inline:nnnn { #1 } { #3 } \l_@@_last_empty_row_int
6776     {

```

We treat the row number ##1 with another loop.

```

6777         \bool_set_false:N \l_tmpa_bool
6778         \int_step_inline:nnnn { #2 } { #4 } \l_@@_last_empty_column_int
6779         {
6780             \@@_test_if_cell_in_a_block:nn { ##1 } { #####1 }
6781             \bool_lazy_or:nnTF
6782                 \l_tmpb_bool
6783                 {
6784                     \cs_if_exist_p:c
6785                         { pgf @ sh @ ns @ \@@_env: - ##1 - #####1 }
6786                 }
6787             { \bool_set_true:N \l_tmpa_bool }
6788             {
6789                 \bool_if:NF \l_tmpa_bool
6790                 {
6791                     \int_set:Nn \l_@@_last_empty_column_int { #####1 }
6792                     \seq_put_right:Nn
6793                         \l_@@_corners_cells_seq
6794                         { ##1 - #####1 }
6795                 }
6796             }
6797         }
6798     }
6799 }

```

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`).

```

6800 \cs_new_protected:Npn \@@_test_if_cell_in_a_block:nn #1 #2
6801 {
6802     \int_set:Nn \l_tmpa_int { #1 }
6803     \int_set:Nn \l_tmpb_int { #2 }
6804     \bool_set_false:N \l_tmpb_bool
6805     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq

```

```

6806     { \@@_test_if_cell_in_block:nnnnnnn \l_tmpa_int \l_tmpb_int ##1 }
6807   }
6808 \cs_set_protected:Npn \@@_test_if_cell_in_block:nnnnnnn #1 #2 #3 #4 #5 #6 #7
6809 {
6810   \int_compare:nNnF { #3 } > { #1 }
6811   {
6812     \int_compare:nNnF { #1 } > { #5 }
6813     {
6814       \int_compare:nNnF { #4 } > { #2 }
6815       {
6816         \int_compare:nNnF { #2 } > { #6 }
6817         { \bool_set_true:N \l_tmpb_bool }
6818       }
6819     }
6820   }
6821 }

```

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.

```

6822 \bool_new:N \l_@@_block_auto_columns_width_bool

```

Up to now, there is only one option available for the environment {NiceMatrixBlock}.

```

6823 \keys_define:nn { NiceMatrix / NiceMatrixBlock }
6824 {
6825   auto-columns-width .code:n =
6826   {
6827     \bool_set_true:N \l_@@_block_auto_columns_width_bool
6828     \dim_gzero_new:N \g_@@_max_cell_width_dim
6829     \bool_set_true:N \l_@@_auto_columns_width_bool
6830   }
6831 }

6832 \NewDocumentEnvironment { NiceMatrixBlock } { ! 0 { } }
6833 {
6834   \int_gincr:N \g_@@_NiceMatrixBlock_int
6835   \dim_zero:N \l_@@_columns_width_dim
6836   \keys_set:nn { NiceMatrix / NiceMatrixBlock } { #1 }
6837   \bool_if:NT \l_@@_block_auto_columns_width_bool
6838   {
6839     \cs_if_exist:cT
6840     { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
6841     {
6842       % is \exp_args:NNe mandatory?
6843       \exp_args:NNe \dim_set:Nn \l_@@_columns_width_dim
6844       {
6845         \use:c
6846         { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
6847       }
6848     }
6849   }
6850 }

```

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).

```

6851 {
6852   \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.

```

6853   { \int_gdecr:N \g_@@_NiceMatrixBlock_int }
6854   {
6855     \bool_if:NT \l_@@_block_auto_columns_width_bool
6856     {
6857       \iow_shipout:Nn \@mainaux \ExplSyntaxOn
6858       \iow_shipout:Nx \@mainaux
6859       {
6860         \cs_gset:cpn
6861         { @@ _ 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.

```

6862         { \dim_eval:n { \g_@@_max_cell_width_dim + \arrayrulewidth } }
6863       }
6864       \iow_shipout:Nn \@mainaux \ExplSyntaxOff
6865     }
6866   }
6867   \ignorespacesafterend
6868 }

```

26 The extra nodes

First, two variants of the functions `\dim_min:nn` and `\dim_max:nn`.

```

6869 \cs_generate_variant:Nn \dim_min:nn { v n }
6870 \cs_generate_variant:Nn \dim_max:nn { v n }

```

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).

```

6871 \cs_new_protected:Npn \@@_create_extra_nodes:
6872 {
6873   \bool_if:nTF \l_@@_medium_nodes_bool
6874   {
6875     \bool_if:NTF \l_@@_large_nodes_bool
6876     \@@_create_medium_and_large_nodes:
6877     \@@_create_medium_nodes:
6878   }
6879   { \bool_if:NT \l_@@_large_nodes_bool \@@_create_large_nodes: }
6880 }

```

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`.

```

6881 \cs_new_protected:Npn \@@_computations_for_medium_nodes:
6882 {
6883   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6884   {
6885     \dim_zero_new:c { l_@@_row_\@@_i: _min_dim }
6886     \dim_set_eq:cN { l_@@_row_\@@_i: _min_dim } \c_max_dim
6887     \dim_zero_new:c { l_@@_row_\@@_i: _max_dim }
6888     \dim_set:cn { l_@@_row_\@@_i: _max_dim } { - \c_max_dim }
6889   }
6890   \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
6891   {
6892     \dim_zero_new:c { l_@@_column_\@@_j: _min_dim }
6893     \dim_set_eq:cN { l_@@_column_\@@_j: _min_dim } \c_max_dim
6894     \dim_zero_new:c { l_@@_column_\@@_j: _max_dim }
6895     \dim_set:cn { l_@@_column_\@@_j: _max_dim } { - \c_max_dim }
6896   }

```

We begin the two nested loops over the rows and the columns of the array.

```

6897   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6898   {
6899     \int_step_variable:nnNn
6900     \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.

```

6901     {
6902       \cs_if_exist:cT
6903       { 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`.

```

6904     {
6905       \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { south-west }
6906       \dim_set:cn { l_@@_row_\@@_i: _min_dim }
6907       { \dim_min:vn { l_@@_row _ \@@_i: _min_dim } \pgf@y }
6908       \seq_if_in:NxF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
6909       {
6910         \dim_set:cn { l_@@_column _ \@@_j: _min_dim }
6911         { \dim_min:vn { l_@@_column _ \@@_j: _min_dim } \pgf@x }
6912       }

```

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`.

```

6913       \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { north-east }
6914       \dim_set:cn { l_@@_row _ \@@_i: _ max_dim }
6915       { \dim_max:vn { l_@@_row _ \@@_i: _ max_dim } \pgf@y }
6916       \seq_if_in:NxF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
6917       {
6918         \dim_set:cn { l_@@_column _ \@@_j: _ max_dim }
6919         { \dim_max:vn { l_@@_column _ \@@_j: _ max_dim } \pgf@x }
6920       }
6921     }
6922   }
6923 }

```

Now, we have to deal with empty rows or empty columns since we don't have created nodes in such rows and columns.

```

6924   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6925   {
6926     \dim_compare:nNnT
6927     { \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } } = \c_max_dim

```

```

6928     {
6929         \@@_qpoint:n { row - \@@_i: - base }
6930         \dim_set:cn { l_@@_row _ \@@_i: _ max _ dim } \pgf@y
6931         \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim } \pgf@y
6932     }
6933 }
6934 \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
6935 {
6936     \dim_compare:nNnT
6937     { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } } = \c_max_dim
6938     {
6939         \@@_qpoint:n { col - \@@_j: }
6940         \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim } \pgf@y
6941         \dim_set:cn { l_@@_column _ \@@_j: _ min _ dim } \pgf@y
6942     }
6943 }
6944 }

```

Here is the command `\@@_create_medium_nodes:`. When this command is used, the “medium nodes” are created.

```

6945 \cs_new_protected:Npn \@@_create_medium_nodes:
6946 {
6947     \pgfpicture
6948     \pgfrememberpicturepositiononpagetrue
6949     \pgf@relevantforpicturesizefalse
6950     \@@_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”.

```

6951     \cs_set_nopar:Npn \l_@@_suffix_tl { -medium }
6952     \@@_create_nodes:
6953     \endpgfpicture
6954 }

```

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:`.

```

6955 \cs_new_protected:Npn \@@_create_large_nodes:
6956 {
6957     \pgfpicture
6958     \pgfrememberpicturepositiononpagetrue
6959     \pgf@relevantforpicturesizefalse
6960     \@@_computations_for_medium_nodes:
6961     \@@_computations_for_large_nodes:
6962     \cs_set_nopar:Npn \l_@@_suffix_tl { - large }
6963     \@@_create_nodes:
6964     \endpgfpicture
6965 }
6966 \cs_new_protected:Npn \@@_create_medium_and_large_nodes:
6967 {
6968     \pgfpicture
6969     \pgfrememberpicturepositiononpagetrue
6970     \pgf@relevantforpicturesizefalse
6971     \@@_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”.

¹⁴If we want to create both, we have to use `\@@_create_medium_and_large_nodes:`

```

6972     \cs_set_nopar:Npn \l_@@_suffix_tl { - medium }
6973     \@@_create_nodes:
6974     \@@_computations_for_large_nodes:
6975     \cs_set_nopar:Npn \l_@@_suffix_tl { - large }
6976     \@@_create_nodes:
6977 \endpgfpicture
6978 }

```

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.

```

6979 \cs_new_protected:Npn \@@_computations_for_large_nodes:
6980 {
6981     \int_set_eq:NN \l_@@_first_row_int \c_one_int
6982     \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`.

```

6983     \int_step_variable:nNn { \c@iRow - 1 } \@@_i:
6984     {
6985         \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim }
6986         {
6987             (
6988                 \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } +
6989                 \dim_use:c { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
6990             )
6991             / 2
6992         }
6993         \dim_set_eq:cc { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
6994         { l_@@_row _ \@@_i: _ min _ dim }
6995     }
6996     \int_step_variable:nNn { \c@jCol - 1 } \@@_j:
6997     {
6998         \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim }
6999         {
7000             (
7001                 \dim_use:c { l_@@_column _ \@@_j: _ max _ dim } +
7002                 \dim_use:c
7003                 { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
7004             )
7005             / 2
7006         }
7007         \dim_set_eq:cc { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
7008         { l_@@_column _ \@@_j: _ max _ dim }
7009     }

```

Here, we have to use `\dim_sub:cn` because of the number 1 in the name.

```

7010     \dim_sub:cn
7011     { l_@@_column _ 1 _ min _ dim }
7012     \l_@@_left_margin_dim
7013     \dim_add:cn
7014     { l_@@_column _ \int_use:N \c@jCol _ max _ dim }
7015     \l_@@_right_margin_dim
7016 }

```

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`).

```

7017 \cs_new_protected:Npn \@@_create_nodes:
7018 {

```

```

7019 \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7020 {
7021     \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
7022     {

```

We draw the rectangular node for the cell ($\@@_i$ - $\@@_j$).

```

7023     \@@_pgf_rect_node:nnnnn
7024     { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7025     { \dim_use:c { l_@@_column_ \@@_j: _min_dim } }
7026     { \dim_use:c { l_@@_row_ \@@_i: _min_dim } }
7027     { \dim_use:c { l_@@_column_ \@@_j: _max_dim } }
7028     { \dim_use:c { l_@@_row_ \@@_i: _max_dim } }
7029     \str_if_empty:NF \l_@@_name_str
7030     {
7031         \pgfnodealias
7032         { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
7033         { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7034     }
7035 }
7036 }

```

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 .

```

7037 \seq_map_pairwise_function:NNN
7038 \g_@@_multicolumn_cells_seq
7039 \g_@@_multicolumn_sizes_seq
7040 \@@_node_for_multicolumn:nn
7041 }

```

```

7042 \cs_new_protected:Npn \@@_extract_coords_values: #1 - #2 \q_stop
7043 {
7044     \cs_set_nopar:Npn \@@_i: { #1 }
7045     \cs_set_nopar:Npn \@@_j: { #2 }
7046 }

```

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”).

```

7047 \cs_new_protected:Npn \@@_node_for_multicolumn:nn #1 #2
7048 {
7049     \@@_extract_coords_values: #1 \q_stop
7050     \@@_pgf_rect_node:nnnnn
7051     { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7052     { \dim_use:c { l_@@_column_ \@@_j: _min_dim } }
7053     { \dim_use:c { l_@@_row_ \@@_i: _min_dim } }
7054     { \dim_use:c { l_@@_column_ \int_eval:n { \@@_j: +#2-1 } _max_dim } }
7055     { \dim_use:c { l_@@_row_ \@@_i: _max_dim } }
7056     \str_if_empty:NF \l_@@_name_str
7057     {
7058         \pgfnodealias
7059         { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
7060         { \int_use:N \g_@@_env_int - \@@_i: - \@@_j: \l_@@_suffix_tl }
7061     }
7062 }

```

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).

```

7063 \keys_define:nn { NiceMatrix / Block / FirstPass }
7064 {
7065   j .code:n = \str_set:Nn \l_@@_hpos_block_str j
7066             \bool_set_true:N \l_@@_p_block_bool ,
7067   j .value_forbidden:n = true ,
7068   l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7069   l .value_forbidden:n = true ,
7070   r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7071   r .value_forbidden:n = true ,
7072   c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7073   c .value_forbidden:n = true ,
7074   L .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7075   L .value_forbidden:n = true ,
7076   R .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7077   R .value_forbidden:n = true ,
7078   C .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7079   C .value_forbidden:n = true ,
7080   t .code:n = \str_set:Nn \l_@@_vpos_block_str t ,
7081   t .value_forbidden:n = true ,
7082   T .code:n = \str_set:Nn \l_@@_vpos_block_str T ,
7083   T .value_forbidden:n = true ,
7084   b .code:n = \str_set:Nn \l_@@_vpos_block_str b ,
7085   b .value_forbidden:n = true ,
7086   B .code:n = \str_set:Nn \l_@@_vpos_block_str B ,
7087   B .value_forbidden:n = true ,
7088   p .code:n = \bool_set_true:N \l_@@_p_block_bool ,
7089   p .value_forbidden:n = true ,
7090   color .code:n =
7091     \@@_color:n { #1 }
7092     \tl_set_rescan:Nnn
7093       \l_@@_draw_tl
7094       { \char_set_catcode_other:N ! }
7095       { #1 } ,
7096   color .value_required:n = true ,
7097   respect-arraystretch .code:n =
7098     \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
7099   respect-arraystretch .value_forbidden:n = true ,
7100 }

```

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.

```

7101 \cs_new_protected:Npn \@@_Block: { \@@_collect_options:n { \@@_Block_i: } }

7102 \NewExpandableDocumentCommand \@@_Block_i: { m m D < > { } +m }
7103 {

```

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).

```

7104   \peek_remove_spaces:n
7105   {
7106     \tl_if_blank:nTF { #2 }
7107     { \@@_Block_ii:nnnnn \c_one_int \c_one_int }
7108     {
7109       \int_compare:nNnTF { \char_value_catcode:n { 45 } } = { 13 }
7110       \@@_Block_i_czech \@@_Block_i
7111       #2 \q_stop
7112     }
7113     { #1 } { #3 } { #4 }

```

```

7114     }
7115 }

```

With the following construction, we extract the values of i and j in the first mandatory argument of the command.

```

7116 \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`.

```

7117 {
7118   \char_set_catcode_active:N -
7119   \cs_new:Npn \@@_Block_i_czech #1-#2 \q_stop { \@@_Block_ii:nnnnn { #1 } { #2 } }
7120 }

```

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.

```

7121 \cs_new_protected:Npn \@@_Block_ii:nnnnn #1 #2 #3 #4 #5
7122 {

```

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).

```

7123   \bool_lazy_or:nnTF
7124     { \tl_if_blank_p:n { #1 } }
7125     { \str_if_eq_p:Vn \c_@@_star_str { #1 } }
7126     { \int_set:Nn \l_tmpa_int { 100 } }
7127     { \int_set:Nn \l_tmpa_int { #1 } }
7128   \bool_lazy_or:nnTF
7129     { \tl_if_blank_p:n { #2 } }
7130     { \str_if_eq_p:Vn \c_@@_star_str { #2 } }
7131     { \int_set:Nn \l_tmpb_int { 100 } }
7132     { \int_set:Nn \l_tmpb_int { #2 } }

```

If the block is mono-column.

```

7133   \int_compare:nNnTF \l_tmpb_int = \c_one_int
7134   {
7135     \tl_if_empty:NNTF \l_@@_hpos_cell_tl
7136       { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_c_str }
7137       { \str_set:NV \l_@@_hpos_block_str \l_@@_hpos_cell_tl }
7138   }
7139   { \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.

```

7140   \keys_set:known:nn { NiceMatrix / Block / FirstPass } { #3 }
7141   \tl_set:Nx \l_tmpa_tl
7142   {
7143     { \int_use:N \c@iRow }
7144     { \int_use:N \c@jCol }
7145     { \int_eval:n { \c@iRow + \l_tmpa_int - 1 } }
7146     { \int_eval:n { \c@jCol + \l_tmpb_int - 1 } }
7147   }

```

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).

```

7148 \bool_set_false:N \l_tmpa_bool
7149 \bool_if:NT \l_@@_amp_in_blocks_bool
7150 { \tl_if_in:nnT { #5 } { & } { \bool_set_true:N \l_tmpa_bool } }
7151 \bool_if:NTF \l_tmpa_bool
7152 { \exp_args:Nee \@@_Block_vii:nnnnn }
7153 {
7154   \bool_if:NTF \l_@@_p_block_bool
7155   { \exp_args:Nee \@@_Block_vi:nnnnn }
7156   {
7157     \bool_if:nTF
7158     {
7159       (
7160         \int_compare_p:nNn \l_tmpa_int = \c_one_int
7161         ||
7162         \int_compare_p:nNn \l_tmpb_int = \c_one_int
7163       )
7164       && ! \tl_if_empty_p:n { #5 }

```

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.

```

7165       && ! \l_@@_X_bool
7166     }
7167     { \exp_args:Nee \@@_Block_iv:nnnnn }
7168     { \exp_args:Nee \@@_Block_v:nnnnn }
7169   }
7170 }
7171 { \l_tmpa_int } { \l_tmpb_int } { #3 } { #4 } { #5 }
7172 }

```

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.

```

7173 \cs_new_protected:Npn \@@_Block_iv:nnnnn #1 #2 #3 #4 #5
7174 {
7175   \int_gincr:N \g_@@_block_box_int
7176   \cs_set_protected_nopar:Npn \diagbox ##1 ##2
7177   {
7178     \tl_gput_right:Nx \g_@@_pre_code_after_tl
7179     {
7180       \@@_actually_diagbox:nnnnnn
7181       { \int_use:N \c_iRow }
7182       { \int_use:N \c_jCol }
7183       { \int_eval:n { \c_iRow + #1 - 1 } }
7184       { \int_eval:n { \c_jCol + #2 - 1 } }
7185       { \g_@@_row_style_tl \exp_not:n { ##1 } }
7186       { \g_@@_row_style_tl \exp_not:n { ##2 } }
7187     }
7188   }
7189   \box_gclear_new:c
7190   { \g_@@_block_box_int \int_use:N \g_@@_block_box_int _box }

```

Now, we will actually compose the content of the `\Block` in a TeX box. *Be careful*: if after the construction of the box, the boolean `\g_@@_rotate_bool` is raised (which means that the command `\rotate` was present in the content of the `\Block`) we will rotate the box but also, maybe, change the position of the baseline!

```

7191 \hbox_gset:cn
7192 { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
7193 {

```

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}`).

```

7194 \tl_if_empty:NTF \l_@@_color_tl
7195 { \int_compare:nNnT { #2 } = \c_one_int \set@color }
7196 { \@@_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`.

```

7197 \int_compare:nNnT { #1 } = \c_one_int
7198 {
7199   \int_if_zero:nTF \c@iRow
7200   \l_@@_code_for_first_row_tl
7201   {
7202     \int_compare:nNnT \c@iRow = \l_@@_last_row_int
7203     \l_@@_code_for_last_row_tl
7204   }
7205   \g_@@_row_style_tl
7206 }

```

The following command will be no-op when `respect-arraystretch` is in force.

```

7207 \@@_reset_arraystretch:
7208 \dim_zero:N \extrarowheight

```

#4 is the optional argument of the command `\Block`, provided with the syntax `<...>`.

```

7209 #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.).

```

7210 \@@_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}`.

```

7211 \bool_if:NTF \l_@@_tabular_bool
7212 {
7213   \bool_lazy_all:nTF
7214   {
7215     { \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`.

```

7216 { ! \dim_compare_p:nNn \l_@@_col_width_dim < \c_zero_dim }
7217 { ! \g_@@_rotate_bool }
7218 }

```

When the block is mono-column in a column with a fixed width (eg `p{3cm}`), we use a `{minipage}`.

```

7219 {
7220   \use:e
7221   {
7222     \exp_not:N \begin { minipage }%
7223     [ \str_lowercase:o \l_@@_vpos_block_str ]
7224     { \l_@@_col_width_dim }

```



```

7225         \str_case:on \l_@@_hpos_block_str
7226         { c \centering r \raggedleft l \raggedright }
7227     }
7228     #5
7229     \end { minipage }
7230 }

```

In the other cases, we use a `{tabular}`.

```

7231     {
7232     \use:e
7233     {
7234         \exp_not:N \begin { tabular }%
7235         [ \str_lowercase:o \l_@@_vpos_block_str ]
7236         { @ { } \l_@@_hpos_block_str @ { } }
7237     }
7238     #5
7239     \end { tabular }
7240 }
7241 }

```

If we are in a mathematical array (`\l_@@_tabular_bool` is false). The composition is always done with an `{array}` (never with a `{minipage}`).

```

7242     {
7243     \c_math_toggle_token
7244     \use:e
7245     {
7246         \exp_not:N \begin { array }%
7247         [ \str_lowercase:o \l_@@_vpos_block_str ]
7248         { @ { } \l_@@_hpos_block_str @ { } }
7249     }
7250     #5
7251     \end { array }
7252     \c_math_toggle_token
7253     }
7254 }

```

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).

```

7255     \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.

```

7256     \int_compare:nNnT { #2 } = \c_one_int
7257     {
7258         \dim_gset:Nn \g_@@_blocks_wd_dim
7259         {
7260             \dim_max:nn
7261             \g_@@_blocks_wd_dim
7262             {
7263                 \box_wd:c
7264                 { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7265             }
7266         }
7267     }

```

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.

```

7268     \int_compare:nNnT { #1 } = \c_one_int
7269     {
7270         \dim_gset:Nn \g_@@_blocks_ht_dim
7271         {
7272             \dim_max:nn
7273             \g_@@_blocks_ht_dim

```

```

7274         {
7275             \box_ht:c
7276             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7277         }
7278     }
7279     \dim_gset:Nn \g_@@_blocks_dp_dim
7280     {
7281         \dim_max:nn
7282         \g_@@_blocks_dp_dim
7283         {
7284             \box_dp:c
7285             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7286         }
7287     }
7288 }
7289 \seq_gput_right:Nx \g_@@_blocks_seq
7290 {
7291     \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.

```

7292     {
7293         \exp_not:n { #3 } ,
7294         \l_@@_hpos_block_str ,

```

Now, we put a key for the vertical alignment.

```

7295         \bool_if:NT \g_@@_rotate_bool
7296         {
7297             \bool_if:NTF \g_@@_rotate_c_bool
7298             { m }
7299             { \int_compare:nNnT \c@iRow = \l_@@_last_row_int T }
7300         }
7301     }
7302     {
7303         \box_use_drop:c
7304         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7305     }
7306 }
7307 \bool_set_false:N \g_@@_rotate_c_bool
7308 }
7309 }

7310 \cs_new:Npn \@@_adjust_hpos_rotate:
7311 {
7312     \bool_if:NT \g_@@_rotate_bool
7313     {
7314         \str_set:Nx \l_@@_hpos_block_str
7315         {
7316             \bool_if:NTF \g_@@_rotate_c_bool
7317             { c }
7318             {
7319                 \str_case:onF \l_@@_vpos_block_str
7320                 { b l B l t r T r }
7321                 { \int_compare:nNnTF \c@iRow = \l_@@_last_row_int r l }
7322             }
7323         }
7324     }
7325 }

```

Despite its name the following command rotates the box of the block *but also does vertical adjustment of the baseline of the block*.

```

7326 \cs_new_protected:Npn \@@_rotate_box_of_block:
7327 {
7328   \box_grotate:cn
7329   { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
7330   { 90 }
7331   \int_compare:nNnT \c@iRow = \l_@@_last_row_int
7332   {
7333     \vbox_gset_top:cn
7334     { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
7335     {
7336       \skip_vertical:n { 0.8 ex }
7337       \box_use:c
7338       { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
7339     }
7340   }
7341   \bool_if:NT \g_@@_rotate_c_bool
7342   {
7343     \hbox_gset:cn
7344     { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
7345     {
7346       \c_math_toggle_token
7347       \vcenter
7348       {
7349         \box_use:c
7350         { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
7351       }
7352       \c_math_toggle_token
7353     }
7354   }
7355 }

```

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.

```

7356 \cs_new_protected:Npn \@@_Block_v:nnnnn #1 #2 #3 #4 #5
7357 {
7358   \seq_gput_right:Nx \g_@@_blocks_seq
7359   {
7360     \l_tmpa_tl
7361     { \exp_not:n { #3 } }
7362     {
7363       \bool_if:NTF \l_@@_tabular_bool
7364       {
7365         \group_begin:

```

The following command will be no-op when `respect-arraystretch` is in force.

```

7366   \@@_reset_arraystretch:
7367   \exp_not:n
7368   {
7369     \dim_zero:N \extrarowheight
7370     #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).

```

7371   \bool_if:NT \c_@@_testphase_table_bool
7372   { \tag_stop:n { table } }

```

```

7373         \use:e
7374         {
7375             \exp_not:N \begin { tabular } [ \l_@@_vpos_block_str ]
7376             { @ { } \l_@@_hpos_block_str @ { } }
7377         }
7378         #5
7379         \end { tabular }
7380     }
7381     \group_end:
7382 }

```

When we are *not* in an environment {NiceTabular} (or similar).

```

7383     {
7384         \group_begin:

```

The following will be no-op when respect-arraystretch is in force.

```

7385         \@@_reset_arraystretch:
7386         \exp_not:n
7387         {
7388             \dim_zero:N \extrarowheight
7389             #4
7390             \c_math_toggle_token
7391             \use:e
7392             {
7393                 \exp_not:N \begin { array } [ \l_@@_vpos_block_str ]
7394                 { @ { } \l_@@_hpos_block_str @ { } }
7395             }
7396             #5
7397             \end { array }
7398             \c_math_toggle_token
7399         }
7400         \group_end:
7401     }
7402 }
7403 }
7404 }

```

The following macro is for the case of a \Block which uses the key p.

```

7405 \cs_new_protected:Npn \@@_Block_vi:nnnnn #1 #2 #3 #4 #5
7406 {
7407     \seq_gput_right:Nx \g_@@_blocks_seq
7408     {
7409         \l_tmpa_tl
7410         { \exp_not:n { #3 } }
7411         {
7412             \group_begin:
7413             \exp_not:n { #4 #5 }
7414             \group_end:
7415         }
7416     }
7417 }

```

The following macro is for the case of a \Block which uses the key p.

```

7418 \cs_new_protected:Npn \@@_Block_vii:nnnnn #1 #2 #3 #4 #5
7419 {
7420     \seq_gput_right:Nx \g_@@_blocks_seq
7421     {
7422         \l_tmpa_tl
7423         { \exp_not:n { #3 } }
7424         { \exp_not:n { #4 #5 } }
7425     }
7426 }

```

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).

```

7427 \keys_define:nn { NiceMatrix / Block / SecondPass }
7428 {
7429   ampersand-in-blocks .bool_set:N = \l_@@_amp_in_blocks_bool ,
7430   ampersand-in-blocks .default:n = true ,
7431   &-in-blocks .meta:n = ampersand-in-blocks ,
7432   tikz .code:n =
7433     \IfPackageLoadedTF { tikz }
7434       { \seq_put_right:Nn \l_@@_tikz_seq { { #1 } } }
7435       { \@@_error:n { tikz~key~without~tikz } } ,
7436   tikz .value_required:n = true ,
7437   fill .code:n =
7438     \tl_set_rescan:Nnn
7439     \l_@@_fill_tl
7440     { \char_set_catcode_other:N ! }
7441     { #1 } ,
7442   fill .value_required:n = true ,
7443   opacity .tl_set:N = \l_@@_opacity_tl ,
7444   opacity .value_required:n = true ,
7445   draw .code:n =
7446     \tl_set_rescan:Nnn
7447     \l_@@_draw_tl
7448     { \char_set_catcode_other:N ! }
7449     { #1 } ,
7450   draw .default:n = default ,
7451   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
7452   rounded-corners .default:n = 4 pt ,
7453   color .code:n =
7454     \@@_color:n { #1 }
7455     \tl_set_rescan:Nnn
7456     \l_@@_draw_tl
7457     { \char_set_catcode_other:N ! }
7458     { #1 } ,
7459   borders .clist_set:N = \l_@@_borders_clist ,
7460   borders .value_required:n = true ,
7461   hvlines .meta:n = { vlines , hlines } ,
7462   vlines .bool_set:N = \l_@@_vlines_block_bool ,
7463   vlines .default:n = true ,
7464   hlines .bool_set:N = \l_@@_hlines_block_bool ,
7465   hlines .default:n = true ,
7466   line-width .dim_set:N = \l_@@_line_width_dim ,
7467   line-width .value_required:n = true ,

```

Some keys have not a property `.value_required:n` (or similar) because they are in `FirstPass`.

```

7468   j .code:n = \str_set:Nn \l_@@_hpos_block_str j
7469             \bool_set_true:N \l_@@_p_block_bool ,
7470   l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7471   r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7472   c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7473   L .code:n = \str_set:Nn \l_@@_hpos_block_str l
7474             \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7475   R .code:n = \str_set:Nn \l_@@_hpos_block_str r
7476             \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7477   C .code:n = \str_set:Nn \l_@@_hpos_block_str c
7478             \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7479   t .code:n = \str_set:Nn \l_@@_vpos_block_str t ,
7480   T .code:n = \str_set:Nn \l_@@_vpos_block_str T ,
7481   b .code:n = \str_set:Nn \l_@@_vpos_block_str b ,
7482   B .code:n = \str_set:Nn \l_@@_vpos_block_str B ,
7483   m .code:n = \str_set:Nn \l_@@_vpos_block_str c ,
7484   m .value_forbidden:n = true ,

```

```

7485     v-center .meta:n = m ,
7486     p .code:n = \bool_set_true:N \l_@@_p_block_bool ,
7487     p .value_forbidden:n = true ,
7488     name .tl_set:N = \l_@@_block_name_str ,
7489     name .value_required:n = true ,
7490     name .initial:n = ,
7491     respect-arraystretch .code:n =
7492       \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
7493     respect-arraystretch .value_forbidden:n = true ,
7494     transparent .bool_set:N = \l_@@_transparent_bool ,
7495     transparent .default:n = true ,
7496     transparent .initial:n = false ,
7497     unknown .code:n = \@@_error:n { Unknown~key~for~Block }
7498   }

```

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.

```

7499 \cs_new_protected:Npn \@@_draw_blocks:
7500 {
7501   \bool_if:NTF \c_@@_tagging_array_bool
7502     { \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign: }
7503     { \cs_set_eq:NN \ialign \@@_old_ialign: }
7504   \seq_map_inline:Nn \g_@@_blocks_seq { \@@_Block_iv:nnnnnn ##1 }
7505 }
7506 \cs_new_protected:Npn \@@_Block_iv:nnnnnn #1 #2 #3 #4 #5 #6
7507 {

```

The integer `\l_@@_last_row_int` will be the last row of the block and `\l_@@_last_col_int` its last column.

```

7508   \int_zero_new:N \l_@@_last_row_int
7509   \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.

```

7510   \int_compare:nNnTF { #3 } > { 99 }
7511     { \int_set_eq:NN \l_@@_last_row_int \c@iRow }
7512     { \int_set:Nn \l_@@_last_row_int { #3 } }
7513   \int_compare:nNnTF { #4 } > { 99 }
7514     { \int_set_eq:NN \l_@@_last_col_int \c@jCol }
7515     { \int_set:Nn \l_@@_last_col_int { #4 } }
7516   \int_compare:nNnTF \l_@@_last_col_int > \g_@@_col_total_int
7517     {
7518       \bool_lazy_and:nnTF
7519         \l_@@_preamble_bool
7520         {
7521           \int_compare_p:n
7522             { \l_@@_last_col_int <= \g_@@_static_num_of_col_int }
7523         }
7524         {
7525           \msg_error:nnnn { nicematrix } { Block-too-large-2 } { #1 } { #2 }
7526           \@@_msg_redirect_name:nn { Block-too-large-2 } { none }
7527           \@@_msg_redirect_name:nn { columns-not-used } { none }
7528         }
7529         { \msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 } }
7530     }
7531   {
7532     \int_compare:nNnTF \l_@@_last_row_int > \g_@@_row_total_int

```

```

7533     { \msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 } }
7534     {
7535         \@@_Block_v:nnVVnn
7536         { #1 }
7537         { #2 }
7538         \l_@@_last_row_int
7539         \l_@@_last_col_int
7540         { #5 }
7541         { #6 }
7542     }
7543 }
7544 }
7545 \cs_generate_variant:Nn \@@_Block_v:nnnnnn { n n V V n n }

```

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

```

7546 \cs_new_protected:Npn \@@_Block_v:nnnnnn #1 #2 #3 #4 #5 #6
7547 {

```

The group is for the keys.

```

7548     \group_begin:
7549     \int_compare:nNnT { #1 } = { #3 }
7550     { \str_set:Nn \l_@@_vpos_block_str { t } }
7551     \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).

```

7552     \tl_if_in:nnT { #6 } { & } { \bool_set_true:N \l_@@_ampersand_bool }
7553     \bool_lazy_and:nnT
7554     \l_@@_vlines_block_bool
7555     { ! \l_@@_ampersand_bool }
7556     {
7557         \tl_gput_right:Nx \g_nicematrix_code_after_tl
7558         {
7559             \@@_vlines_block:nnn
7560             { \exp_not:n { #5 } }
7561             { #1 - #2 }
7562             { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7563         }
7564     }
7565     \bool_if:NT \l_@@_hlines_block_bool
7566     {
7567         \tl_gput_right:Nx \g_nicematrix_code_after_tl
7568         {
7569             \@@_hlines_block:nnn
7570             { \exp_not:n { #5 } }
7571             { #1 - #2 }
7572             { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7573         }
7574     }
7575     \bool_if:NF \l_@@_transparent_bool
7576     {
7577         \bool_lazy_and:nnF \l_@@_vlines_block_bool \l_@@_hlines_block_bool
7578         {

```

The sequence of the positions of the blocks (excepted the blocks with the key `hlines`) will be used when drawing the rules (in fact, there is also the `\multicolumn` and the `\diagbox` in that sequence).

```

7579         \seq_gput_left:Nx \g_@@_pos_of_blocks_seq
7580         { { #1 } { #2 } { #3 } { #4 } { \l_@@_block_name_str } }
7581     }
7582 }

```

```

7583 \tl_if_empty:NF \l_@@_draw_tl
7584 {
7585   \bool_lazy_or:nnT \l_@@_hlines_block_bool \l_@@_vlines_block_bool
7586   { \@@_error:n { hlines~with~color } }
7587   \tl_gput_right:Nx \g_nicematrix_code_after_tl
7588   {
7589     \@@_stroke_block:nnn

```

#5 are the options

```

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     \seq_gput_right:Nn \g_@@_pos_of_stroken_blocks_seq
7595     { { #1 } { #2 } { #3 } { #4 } }
7596   }
7597   \clist_if_empty:NF \l_@@_borders_clist
7598   {
7599     \tl_gput_right:Nx \g_nicematrix_code_after_tl
7600     {
7601       \@@_stroke_borders_block:nnn
7602       { \exp_not:n { #5 } }
7603       { #1 - #2 }
7604       { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7605     }
7606   }
7607   \tl_if_empty:NF \l_@@_fill_tl
7608   {
7609     \tl_if_empty:NF \l_@@_opacity_tl
7610     {
7611       \tl_if_head_eq_meaning:nNTF \l_@@_fill_tl [
7612         {
7613           \tl_set:Nx \l_@@_fill_tl
7614           {
7615             [ opacity = \l_@@_opacity_tl ,
7616             \tl_tail:o \l_@@_fill_tl
7617           ]
7618         }
7619         {
7620           \tl_set:Nx \l_@@_fill_tl
7621           { [ opacity = \l_@@_opacity_tl ] { \l_@@_fill_tl } }
7622         }
7623       ]
7624       \tl_gput_right:Nx \g_@@_pre_code_before_tl
7625       {
7626         \exp_not:N \roundedrectanglecolor
7627         \exp_args:No \tl_if_head_eq_meaning:nNTF \l_@@_fill_tl [
7628           { \l_@@_fill_tl }
7629           { { \l_@@_fill_tl } }
7630           { #1 - #2 }
7631           { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7632           { \dim_use:N \l_@@_rounded_corners_dim }
7633         ]
7634       }
7635       \seq_if_empty:NF \l_@@_tikz_seq
7636       {
7637         \tl_gput_right:Nx \g_nicematrix_code_before_tl
7638         {
7639           \@@_block_tikz:nnnnn
7640           { #1 }
7641           { #2 }
7642           { \int_use:N \l_@@_last_row_int }
7643           { \int_use:N \l_@@_last_col_int }

```



```

7644         { \seq_use:Nn \l_@@_tikz_seq { , } }
7645     }
7646 }

7647 \cs_set_protected_nopar:Npn \diagbox ##1 ##2
7648 {
7649     \tl_gput_right:Nx \g_@@_pre_code_after_tl
7650     {
7651         \@@_actually_diagbox:nnnnnn
7652         { #1 }
7653         { #2 }
7654         { \int_use:N \l_@@_last_row_int }
7655         { \int_use:N \l_@@_last_col_int }
7656         { \exp_not:n { ##1 } } { \exp_not:n { ##2 } }
7657     }
7658 }

```

Let's consider the following `{NiceTabular}`. Because of the instruction `!\hspace{1cm}` in the preamble which increases the space between the columns (by adding, in fact, that space to the previous column, that is to say the second column of the tabular), we will create *two* nodes relative to the block: the node `1-1-block` and the node `1-1-block-short`.

```

\begin{NiceTabular}{cc!\hspace{1cm}}c}
\Block{2-2}{our block} &      & one      & \\
                        &      & two      & \\
three                  & four & five     & \\
six                    & seven & eight    & \\
\end{NiceTabular}

```

We highlight the node `1-1-block`

our block		one
		two
three	four	five
six	seven	eight

We highlight the node `1-1-block-short`

our block		one
		two
three	four	five
six	seven	eight

The construction of the node corresponding to the merged cells.

```

7659 \pgfpicture
7660 \pgfrememberpicturepositiononpagetrue
7661 \pgf@relevantforpicturesizefalse
7662 \@@_qpoint:n { row - #1 }
7663 \dim_set_eq:NN \l_tmpa_dim \pgf@y
7664 \@@_qpoint:n { col - #2 }
7665 \dim_set_eq:NN \l_tmpb_dim \pgf@x
7666 \@@_qpoint:n { row - \int_eval:n { \l_@@_last_row_int + 1 } }
7667 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7668 \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7669 \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.

```

7670 \@@_pgf_rect_node:nnnnn
7671 { \@@_env: - #1 - #2 - block }
7672 \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
7673 \str_if_empty:NF \l_@@_block_name_str
7674 {
7675     \pgfnodealias
7676     { \@@_env: - \l_@@_block_name_str }
7677     { \@@_env: - #1 - #2 - block }
7678     \str_if_empty:NF \l_@@_name_str
7679     {

```

```

7680         \pgfnodealias
7681         { \l_@@_name_str - \l_@@_block_name_str }
7682         { \@@_env: - #1 - #2 - block }
7683     }
7684 }

```

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.

```

7685     \bool_if:NF \l_@@_hpos_of_block_cap_bool
7686     {
7687         \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.

```

7688         \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
7689         {

```

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.

```

7690             \cs_if_exist:cT
7691             { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
7692             {
7693                 \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
7694                 {
7695                     \pgfpointanchor { \@@_env: - ##1 - #2 } { west }
7696                     \dim_set:Nn \l_tmpb_dim { \dim_min:nn \l_tmpb_dim \pgf@x }
7697                 }
7698             }
7699         }

```

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.

```

7700         \dim_compare:nNnT \l_tmpb_dim = \c_max_dim
7701         {
7702             \@@_qpoint:n { col - #2 }
7703             \dim_set_eq:NN \l_tmpb_dim \pgf@x
7704         }
7705     \dim_set:Nn \l_@@_tmpd_dim { - \c_max_dim }
7706     \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
7707     {
7708         \cs_if_exist:cT
7709         { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
7710         {
7711             \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
7712             {
7713                 \pgfpointanchor
7714                 { \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
7715                 { east }
7716                 \dim_set:Nn \l_@@_tmpd_dim { \dim_max:nn \l_@@_tmpd_dim \pgf@x }
7717             }
7718         }
7719     }
7720     \dim_compare:nNnT \l_@@_tmpd_dim = { - \c_max_dim }
7721     {
7722         \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7723         \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
7724     }
7725     \@@_pgf_rect_node:nnnnn
7726     { \@@_env: - #1 - #2 - block - short }
7727     \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
7728 }

```

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.

```

7729 \bool_if:NT \l_@@_medium_nodes_bool
7730 {
7731   \@@_pgf_rect_node:nnn
7732   { \@@_env: - #1 - #2 - block - medium }
7733   { \pgfpointanchor { \@@_env: - #1 - #2 - medium } { north-west } }
7734   {
7735     \pgfpointanchor
7736     { \@@_env:
7737       - \int_use:N \l_@@_last_row_int
7738       - \int_use:N \l_@@_last_col_int - medium
7739     }
7740     { south-east }
7741   }
7742 }
7743 \endpgfpicture

```

```

7744 \bool_if:NTF \l_@@_ampersand_bool
7745 {
7746   \seq_set_split:Nnn \l_tmpa_seq { & } { #6 }
7747   \int_zero_new:N \l_@@_split_int
7748   \int_set:Nn \l_@@_split_int { \seq_count:N \l_tmpa_seq }
7749   \pgfpicture
7750   \pgfrememberpicturepositiononpagetrue
7751   \pgf@relevantforpicturesizefalse
7752   \@@_qpoint:n { row - #1 }
7753   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7754   \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
7755   \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
7756   \@@_qpoint:n { col - #2 }
7757   \dim_set_eq:NN \l_tmpa_dim \pgf@x
7758   \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
7759   \dim_set:Nn \l_tmpb_dim
7760   { ( \pgf@x - \l_tmpa_dim ) / \int_use:N \l_@@_split_int }
7761   \bool_lazy_or:nnT
7762   \l_@@_vlines_block_bool
7763   { \tl_if_eq_p:NN \l_@@_vlines_clist \c_@@_all_tl }
7764   {
7765     \int_step_inline:nn { \l_@@_split_int - 1 }
7766     {
7767       \pgfpathmoveto
7768       {
7769         \pgfpoint
7770         { \l_tmpa_dim + ##1 \l_tmpb_dim }
7771         \l_@@_tmpc_dim
7772       }
7773       \pgfpathlineto
7774       {
7775         \pgfpoint
7776         { \l_tmpa_dim + ##1 \l_tmpb_dim }
7777         \l_@@_tmpd_dim
7778       }
7779       \CT@arc@
7780       \pgfsetlinewidth { 1.1 \arrayrulewidth }
7781       \pgfsetrectcap
7782       \pgfusepathqstroke
7783     }
7784   }
7785   \@@_qpoint:n { row - #1 - base }
7786   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7787   \int_step_inline:nn \l_@@_split_int
7788   {

```

```

7789 \group_begin:
7790 \dim_set:Nn \col@sep
7791 { \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
7792 \pgftransformshift
7793 {
7794   \pgfpoint
7795   {
7796     \str_case:on \l_@@_hpos_block_str
7797     {
7798       l { \l_tmpa_dim + ##1 \l_tmpb_dim - \l_tmpb_dim + \col@sep }
7799       c { \l_tmpa_dim + ##1 \l_tmpb_dim - 0.5 \l_tmpb_dim }
7800       r { \l_tmpa_dim + ##1 \l_tmpb_dim - \col@sep }
7801     }
7802   }
7803   { \l_@@_tmpc_dim }
7804 }
7805 \pgfset
7806 {
7807   inner~xsep = \c_zero_dim ,
7808   inner~ysep = \c_zero_dim
7809 }
7810 \pgfnode
7811 { rectangle }
7812 {
7813   \str_case:on \l_@@_hpos_block_str
7814   {
7815     c { base }
7816     l { base~west }
7817     r { base~east }
7818   }
7819 }
7820 { \seq_item:Nn \l_tmpa_seq { ##1 } } { } { }
7821 \group_end:
7822 }
7823 \endpgfpicture
7824 }
7825 {
7826   \bool_if:NTF \l_@@_p_block_bool
7827   {

```

When the final user has used the key p, we have to compute the width.

```

7828 \pgfpicture
7829 \pgfrememberpicturepositiononpagetrue
7830 \pgf@relevantforpicturesizefalse
7831 \bool_if:NTF \l_@@_hpos_of_block_cap_bool
7832 {
7833   \@@_qpoint:n { col - #2 }
7834   \dim_gset_eq:NN \g_tmpa_dim \pgf@x
7835   \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7836 }
7837 {
7838   \pgfpointanchor { \@@_env: - #1 - #2 - block - short } { west }
7839   \dim_gset_eq:NN \g_tmpa_dim \pgf@x
7840   \pgfpointanchor { \@@_env: - #1 - #2 - block - short } { east }
7841 }
7842 \dim_gset:Nn \g_tmpb_dim { \pgf@x - \g_tmpa_dim }
7843 \endpgfpicture
7844 \hbox_set:Nn \l_@@_cell_box
7845 {
7846   \begin { minipage } [ \str_lowercase:o \l_@@_vpos_block_str ]
7847   { \g_tmpb_dim }
7848   \str_case:on \l_@@_hpos_block_str
7849   { c \centering r \raggedleft l \raggedright j { } }
7850   #6

```

```

7851         \end { minipage }
7852     }
7853 }
7854 { \hbox_set:Nn \l_@@_cell_box { \set@color #6 } }
7855 \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:

```

Now, we will put the label of the block.

```

7856 \pgfpicture
7857 \pgfrememberpicturepositiononpagetrue
7858 \pgf@relevantforpicturesizefalse
7859 \bool_lazy_any:nTF
7860 {
7861     { \str_if_eq_p:on \l_@@_vpos_block_str { c } }
7862     { \str_if_eq_p:on \l_@@_vpos_block_str { T } }
7863     { \str_if_eq_p:on \l_@@_vpos_block_str { B } }
7864 }
7865 {

```

If we are in the first column, we must put the block as if it was with the key r.

```

7866 \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.

```

7867 \bool_if:nT \g_@@_last_col_found_bool
7868 {
7869     \int_compare:nNnT { #2 } = \g_@@_col_total_int
7870     { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_l_str }
7871 }

```

`\l_tmpa_tl` will contain the anchor of the PGF node which will be used.

```

7872 \tl_set:Nx \l_tmpa_tl
7873 {
7874     \str_case:on \l_@@_vpos_block_str
7875     {
7876         c {
7877             \str_case:on \l_@@_hpos_block_str
7878             {
7879                 c { center }
7880                 l { west }
7881                 r { east }
7882                 j { center }
7883             }
7884         }
7885     }
7886     T {
7887         \str_case:on \l_@@_hpos_block_str
7888         {
7889             c { north }
7890             l { north-west }
7891             r { north-east }
7892             j { north }
7893         }
7894     }
7895 }
7896 B {
7897     \str_case:on \l_@@_hpos_block_str
7898     {
7899         c { south }
7900         l { south-west }
7901         r { south-east }
7902         j { south }
7903     }
7904 }
7905 }

```

```

7906     }
7907   }
7908   \pgftransformshift
7909   {
7910     \pgfpointanchor
7911     {
7912       \@@_env: - #1 - #2 - block
7913       \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
7914     }
7915     { \l_tmpa_tl }
7916   }
7917   \pgfset
7918   {
7919     inner-xsep = \c_zero_dim ,
7920     inner-ysep = \c_zero_dim
7921   }
7922   \pgfnode
7923   { rectangle }
7924   { \l_tmpa_tl }
7925   { \box_use_drop:N \l_@@_cell_box } { } { }
7926 }

```

End of the case when `\l_@@_vpos_block_str` is equal to c, T or B. Now, the other cases.

```

7927 {
7928   \pgfextracty \l_tmpa_dim
7929   {
7930     \@@_qpoint:n
7931     {
7932       row - \str_if_eq:onTF \l_@@_vpos_block_str { b } { #3 } { #1 }
7933       - base
7934     }
7935   }
7936   \dim_sub:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }

```

We retrieve (in `\pgf@x`) the x -value of the center of the block.

```

7937   \pgfpointanchor
7938   {
7939     \@@_env: - #1 - #2 - block
7940     \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
7941   }
7942   {
7943     \str_case:on \l_@@_hpos_block_str
7944     {
7945       c { center }
7946       l { west }
7947       r { east }
7948       j { center }
7949     }
7950   }

```

We put the label of the block which has been composed in `\l_@@_cell_box`.

```

7951   \pgftransformshift { \pgfpoint \pgf@x \l_tmpa_dim }
7952   \pgfset { inner-sep = \c_zero_dim }
7953   \pgfnode
7954   { rectangle }
7955   {
7956     \str_case:on \l_@@_hpos_block_str
7957     {
7958       c { base }
7959       l { base~west }
7960       r { base~east }
7961       j { base }
7962     }
7963   }

```

```

7964         { \box_use_drop:N \l_@@_cell_box } { } { }
7965     }
7966 \endpgfpicture
7967 }
7968 \group_end:
7969 }

```

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).

```

7970 \cs_new_protected:Npn \@@_stroke_block:nnn #1 #2 #3
7971 {
7972     \group_begin:
7973     \tl_clear:N \l_@@_draw_tl
7974     \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
7975     \keys_set_known:nn { NiceMatrix / BlockStroke } { #1 }
7976     \pgfpicture
7977     \pgfrememberpicturepositiononpagetrue
7978     \pgf@relevantforpicturesizefalse
7979     \tl_if_empty:NF \l_@@_draw_tl
7980     {

```

If the user has used the key `color` of the command `\Block` without value, the color fixed by `\arrayrulecolor` is used.

```

7981         \tl_if_eq:NNTF \l_@@_draw_tl \c_@@_default_tl
7982         { \CT@arc@ }
7983         { \@@_color:o \l_@@_draw_tl }
7984     }
7985 \pgfsetcornersarced
7986 {
7987     \pgfpoint
7988     { \l_@@_rounded_corners_dim }
7989     { \l_@@_rounded_corners_dim }
7990 }
7991 \@@_cut_on_hyphen:w #2 \q_stop
7992 \int_compare:nNnF \l_tmpa_tl > \c@iRow
7993 {
7994     \int_compare:nNnF \l_tmpb_tl > \c@jCol
7995     {
7996         \@@_qpoint:n { row - \l_tmpa_tl }
7997         \dim_set_eq:NN \l_tmpb_dim \pgf@y
7998         \@@_qpoint:n { col - \l_tmpb_tl }
7999         \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
8000         \@@_cut_on_hyphen:w #3 \q_stop
8001         \int_compare:nNnT \l_tmpa_tl > \c@iRow
8002         { \tl_set:No \l_tmpa_tl { \int_use:N \c@iRow } }
8003         \int_compare:nNnT \l_tmpb_tl > \c@jCol
8004         { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
8005         \@@_qpoint:n { row - \int_eval:n { \l_tmpa_tl + 1 } }
8006         \dim_set_eq:NN \l_tmpa_dim \pgf@y
8007         \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
8008         \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
8009         \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
8010         \pgfpathrectanglecorners
8011         { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
8012         { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
8013         \dim_compare:nNnTF \l_@@_rounded_corners_dim = \c_zero_dim
8014         { \pgfusepathqstroke }
8015         { \pgfusepath { stroke } }
8016     }
8017 }
8018 \endpgfpicture
8019 \group_end:

```

```
8020 }
```

Here is the set of keys for the command `\@@_stroke_block:nnn`.

```
8021 \keys_define:nn { NiceMatrix / BlockStroke }
8022 {
8023   color .tl_set:N = \l_@@_draw_tl ,
8024   draw .code:n =
8025     \exp_args:Ne \tl_if_empty:nF { #1 } { \tl_set:Nn \l_@@_draw_tl { #1 } } ,
8026   draw .default:n = default ,
8027   line-width .dim_set:N = \l_@@_line_width_dim ,
8028   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
8029   rounded-corners .default:n = 4 pt
8030 }
```

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).

```
8031 \cs_new_protected:Npn \@@_vlines_block:nnn #1 #2 #3
8032 {
8033   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8034   \keys_set_known:nn { NiceMatrix / BlockBorders } { #1 }
8035   \@@_cut_on_hyphen:w #2 \q_stop
8036   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
8037   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
8038   \@@_cut_on_hyphen:w #3 \q_stop
8039   \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
8040   \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
8041   \int_step_inline:nnn \l_@@_tmpd_tl \l_tmpb_tl
8042   {
8043     \use:e
8044     {
8045       \@@_vline:n
8046       {
8047         position = ##1 ,
8048         start = \l_@@_tmpc_tl ,
8049         end = \int_eval:n { \l_tmpa_tl - 1 } ,
8050         total-width = \dim_use:N \l_@@_line_width_dim
8051       }
8052     }
8053   }
8054 }
8055 \cs_new_protected:Npn \@@_hlines_block:nnn #1 #2 #3
8056 {
8057   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8058   \keys_set_known:nn { NiceMatrix / BlockBorders } { #1 }
8059   \@@_cut_on_hyphen:w #2 \q_stop
8060   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
8061   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
8062   \@@_cut_on_hyphen:w #3 \q_stop
8063   \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
8064   \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
8065   \int_step_inline:nnn \l_@@_tmpc_tl \l_tmpa_tl
8066   {
8067     \use:e
8068     {
8069       \@@_hline:n
8070       {
8071         position = ##1 ,
8072         start = \l_@@_tmpd_tl ,
8073         end = \int_eval:n { \l_tmpb_tl - 1 } ,
8074         total-width = \dim_use:N \l_@@_line_width_dim
8075       }
8076     }
8077   }
```



```

8077     }
8078 }

```

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).

```

8079 \cs_new_protected:Npn \@@_stroke_borders_block:nnn #1 #2 #3
8080 {
8081   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8082   \keys_set_known:nn { NiceMatrix / BlockBorders } { #1 }
8083   \dim_compare:nNnTF \l_@@_rounded_corners_dim > \c_zero_dim
8084     { \@@_error:n { borders~forbidden } }
8085     {
8086       \tl_clear_new:N \l_@@_borders_tikz_tl
8087       \keys_set:nV
8088         { NiceMatrix / OnlyForTikzInBorders }
8089         \l_@@_borders_clist
8090       \@@_cut_on_hyphen:w #2 \q_stop
8091       \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
8092       \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
8093       \@@_cut_on_hyphen:w #3 \q_stop
8094       \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
8095       \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
8096       \@@_stroke_borders_block_i:
8097     }
8098 }
8099 \hook_gput_code:nnn { begindocument } { . }
8100 {
8101   \cs_new_protected:Npx \@@_stroke_borders_block_i:
8102   {
8103     \c_@@_pgfortikzpicture_tl
8104     \@@_stroke_borders_block_ii:
8105     \c_@@_endpgfortikzpicture_tl
8106   }
8107 }
8108 \cs_new_protected:Npn \@@_stroke_borders_block_ii:
8109 {
8110   \pgfrememberpicturerepositiononpagetrue
8111   \pgf@relevantforpicturesizefalse
8112   \CT@arc@
8113   \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
8114   \clist_if_in:NnT \l_@@_borders_clist { right }
8115     { \@@_stroke_vertical:n \l_tmpb_tl }
8116   \clist_if_in:NnT \l_@@_borders_clist { left }
8117     { \@@_stroke_vertical:n \l_@@_tmpd_tl }
8118   \clist_if_in:NnT \l_@@_borders_clist { bottom }
8119     { \@@_stroke_horizontal:n \l_tmpa_tl }
8120   \clist_if_in:NnT \l_@@_borders_clist { top }
8121     { \@@_stroke_horizontal:n \l_@@_tmpc_tl }
8122 }
8123 \keys_define:nn { NiceMatrix / OnlyForTikzInBorders }
8124 {
8125   tikz .code:n =
8126     \cs_if_exist:NTF \tikzpicture
8127       { \tl_set:Nn \l_@@_borders_tikz_tl { #1 } }
8128       { \@@_error:n { tikz~in~borders~without~tikz } } ,
8129   tikz .value_required:n = true ,
8130   top .code:n = ,
8131   bottom .code:n = ,
8132   left .code:n = ,
8133   right .code:n = ,
8134   unknown .code:n = \@@_error:n { bad~border }

```

```
8135 }
```

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).

```
8136 \cs_new_protected:Npn \@@_stroke_vertical:n #1
8137 {
8138   \@@_qpoint:n \l_@@_tmpc_tl
8139   \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
8140   \@@_qpoint:n \l_tmpa_tl
8141   \dim_set:Nn \l_@@_tmpc_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
8142   \@@_qpoint:n { #1 }
8143   \tl_if_empty:NTF \l_@@_borders_tikz_tl
8144   {
8145     \pgfpathmoveto { \pgfpoint \pgf@x \l_tmpb_dim }
8146     \pgfpathlineto { \pgfpoint \pgf@x \l_@@_tmpc_dim }
8147     \pgfusepathqstroke
8148   }
8149   {
8150     \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
8151     ( \pgf@x , \l_tmpb_dim ) -- ( \pgf@x , \l_@@_tmpc_dim ) ;
8152   }
8153 }
```

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).

```
8154 \cs_new_protected:Npn \@@_stroke_horizontal:n #1
8155 {
8156   \@@_qpoint:n \l_@@_tmpd_tl
8157   \clist_if_in:NnTF \l_@@_borders_clist { left }
8158   { \dim_set:Nn \l_tmpa_dim { \pgf@x - 0.5 \l_@@_line_width_dim } }
8159   { \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \l_@@_line_width_dim } }
8160   \@@_qpoint:n \l_tmpb_tl
8161   \dim_set:Nn \l_tmpb_dim { \pgf@x + 0.5 \l_@@_line_width_dim }
8162   \@@_qpoint:n { #1 }
8163   \tl_if_empty:NTF \l_@@_borders_tikz_tl
8164   {
8165     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }
8166     \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
8167     \pgfusepathqstroke
8168   }
8169   {
8170     \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
8171     ( \l_tmpa_dim , \pgf@y ) -- ( \l_tmpb_dim , \pgf@y ) ;
8172   }
8173 }
```

Here is the set of keys for the command \@@_stroke_borders_block:nnn.

```
8174 \keys_define:nn { NiceMatrix / BlockBorders }
8175 {
8176   borders .clist_set:N = \l_@@_borders_clist ,
8177   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
8178   rounded-corners .default:n = 4 pt ,
8179   line-width .dim_set:N = \l_@@_line_width_dim
8180 }
```

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.

```
8181 \cs_new_protected:Npn \@@_block_tikz:nnnnn #1 #2 #3 #4 #5
```

```

8182 {
8183   \begin { tikzpicture }
8184   \@@_clip_with_rounded_corners:
8185   \clist_map_inline:nn { #5 }
8186   {
8187     \keys_set_known:nnN { NiceMatrix / SpecialOffset } { ##1 } \l_tmpa_tl
8188     \use:e { \exp_not:N \path [ \l_tmpa_tl ] }
8189     (
8190       [
8191         xshift = \dim_use:N \l_@@_offset_dim ,
8192         yshift = - \dim_use:N \l_@@_offset_dim
8193       ]
8194       #1 -| #2
8195     )
8196     rectangle
8197     (
8198       [
8199         xshift = - \dim_use:N \l_@@_offset_dim ,
8200         yshift = \dim_use:N \l_@@_offset_dim
8201       ]
8202       \int_eval:n { #3 + 1 } -| \int_eval:n { #4 + 1 }
8203     ) ;
8204   }
8205   \end { tikzpicture }
8206 }
8207 \cs_generate_variant:Nn \@@_block_tikz:nnnnn { n n n n V }

8208 \keys_define:nn { NiceMatrix / SpecialOffset }
8209 { offset .dim_set:N = \l_@@_offset_dim }

```

28 How to draw the dotted lines transparently

```

8210 \cs_set_protected:Npn \@@_renew_matrix:
8211 {
8212   \RenewDocumentEnvironment { pmatrix } { } {
8213     { \pNiceMatrix }
8214     { \endpNiceMatrix }
8215   }
8216   \RenewDocumentEnvironment { vmatrix } { } {
8217     { \vNiceMatrix }
8218     { \endvNiceMatrix }
8219   }
8220   \RenewDocumentEnvironment { Vmatrix } { } {
8221     { \VNiceMatrix }
8222     { \endVNiceMatrix }
8223   }
8224   \RenewDocumentEnvironment { bmatrix } { } {
8225     { \bNiceMatrix }
8226     { \endbNiceMatrix }
8227   }
8228   \RenewDocumentEnvironment { Bmatrix } { } {
8229     { \BNiceMatrix }
8230     { \endBNiceMatrix }
8231   }
8232 }

```

29 Automatic arrays

We will extract some keys and pass the other keys to the environment `{NiceArrayWithDelims}`.

```

8228 \keys_define:nn { NiceMatrix / Auto }
8229 {
8230   columns-type .tl_set:N = \l_@@_columns_type_tl ,
8231   columns-type .value_required:n = true ,

```

```

8232 l .meta:n = { columns-type = l } ,
8233 r .meta:n = { columns-type = r } ,
8234 c .meta:n = { columns-type = c } ,
8235 delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8236 delimiters / color .value_required:n = true ,
8237 delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
8238 delimiters / max-width .default:n = true ,
8239 delimiters .code:n = \keys_set:nn { NiceMatrix / delimiters } { #1 } ,
8240 delimiters .value_required:n = true ,
8241 rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
8242 rounded-corners .default:n = 4 pt
8243 }

8244 \NewDocumentCommand \AutoNiceMatrixWithDelims
8245 { m m O { } } > { \SplitArgument { 1 } { - } } m O { } m ! O { } }
8246 { \@@_auto_nice_matrix:nnnnnn { #1 } { #2 } #4 { #6 } { #3 , #5 , #7 } }

8247 \cs_new_protected:Npn \@@_auto_nice_matrix:nnnnnn #1 #2 #3 #4 #5 #6
8248 {

```

The group is for the protection of the keys.

```

8249 \group_begin:
8250 \keys_set_known:nnN { NiceMatrix / Auto } { #6 } \l_tmpa_tl
8251 \use:e
8252 {
8253 \exp_not:N \begin { NiceArrayWithDelims } { #1 } { #2 }
8254 { * { #4 } { \exp_not:o \l_@@_columns_type_tl } }
8255 [ \exp_not:o \l_tmpa_tl ]
8256 }
8257 \int_if_zero:nT \l_@@_first_row_int
8258 {
8259 \int_if_zero:nT \l_@@_first_col_int { & }
8260 \prg_replicate:nn { #4 - 1 } { & }
8261 \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
8262 }
8263 \prg_replicate:nn { #3 }
8264 {
8265 \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).

```

8266 \prg_replicate:nn { #4 - 1 } { { } #5 & } #5
8267 \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
8268 }
8269 \int_compare:nNnT \l_@@_last_row_int > { -2 }
8270 {
8271 \int_if_zero:nT \l_@@_first_col_int { & }
8272 \prg_replicate:nn { #4 - 1 } { & }
8273 \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
8274 }
8275 \end { NiceArrayWithDelims }
8276 \group_end:
8277 }

8278 \cs_set_protected:Npn \@@_define_com:nnn #1 #2 #3
8279 {
8280 \cs_set_protected:cpn { #1 AutoNiceMatrix }
8281 {
8282 \bool_gset_true:N \g_@@_delims_bool
8283 \str_gset:Nx \g_@@_name_env_str { #1 AutoNiceMatrix }
8284 \AutoNiceMatrixWithDelims { #2 } { #3 }
8285 }
8286 }

```

```

8287 \@@_define_com:nnn p ( )
8288 \@@_define_com:nnn b [ ]
8289 \@@_define_com:nnn v | |
8290 \@@_define_com:nnn V \ | \ |
8291 \@@_define_com:nnn B \{ \}

```

We define also a command `\AutoNiceMatrix` similar to the environment `{NiceMatrix}`.

```

8292 \NewDocumentCommand \AutoNiceMatrix { O { } m O { } m ! O { } }
8293 {
8294   \group_begin:
8295   \bool_gset_false:N \g_@@_delims_bool
8296   \AutoNiceMatrixWithDelims . . { #2 } { #4 } [ #1 , #3 , #5 ]
8297   \group_end:
8298 }

```

30 The redefinition of the command `\dotfill`

```

8299 \cs_set_eq:NN \@@_old_dotfill \dotfill
8300 \cs_new_protected:Npn \@@_dotfill:
8301 {

```

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}`).

```

8302   \@@_old_dotfill
8303   \tl_gput_right:Nn \g_@@_cell_after_hook_tl \@@_dotfill_i:
8304 }

```

Now, if the box is not empty (unfortunately, we can’t actually test whether the box is empty and that’s why we only consider it’s width), we insert `\@@_dotfill` (which is the saved version of `\dotfill`) in the cell of the array, and it will extend, since it is no longer in `\l_@@_cell_box`.

```

8305 \cs_new_protected:Npn \@@_dotfill_i:
8306 { \dim_compare:nNtT { \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.

```

8307 \cs_new_protected:Npn \@@_diagbox:nn #1 #2
8308 {
8309   \tl_gput_right:Nx \g_@@_pre_code_after_tl
8310   {
8311     \@@_actually_diagbox:nnnnnn
8312     { \int_use:N \c@iRow }
8313     { \int_use:N \c@jCol }
8314     { \int_use:N \c@iRow }
8315     { \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.

```

8316     { \g_@@_row_style_tl \exp_not:n { #1 } }
8317     { \g_@@_row_style_tl \exp_not:n { #2 } }
8318   }

```

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`.

```

8319   \seq_gput_right:Nx \g_@@_pos_of_blocks_seq
8320   {
8321     { \int_use:N \c@iRow }
8322     { \int_use:N \c@jCol }
8323     { \int_use:N \c@iRow }
8324     { \int_use:N \c@jCol }

```

The last argument is for the name of the block.

```

8325     { }
8326   }
8327 }

```

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.

```

8328 \cs_new_protected:Npn \@@_actually_diagbox:nnnnnn #1 #2 #3 #4 #5 #6
8329 {
8330   \pgfpicture
8331   \pgf@relevantforpicturesizefalse
8332   \pgfrememberpicturepositiononpagetrue
8333   \@@_qpoint:n { row - #1 }
8334   \dim_set_eq:NN \l_tmpa_dim \pgf@y
8335   \@@_qpoint:n { col - #2 }
8336   \dim_set_eq:NN \l_tmpb_dim \pgf@x
8337   \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
8338   \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
8339   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8340   \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
8341   \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
8342   \pgfpathlineto { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
8343   {

```

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.

```

8344   \CT@arc@
8345   \pgfsetroundcap
8346   \pgfusepathqstroke
8347 }
8348 \pgfset { inner~sep = 1 pt }
8349 \pgfscope
8350 \pgftransformshift { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
8351 \pgfnode { rectangle } { south~west }
8352 {
8353   \begin { minipage } { 20 cm }
8354   \@@_math_toggle: #5 \@@_math_toggle:
8355   \end { minipage }
8356 }
8357 { }
8358 { }
8359 \endpgfscope
8360 \pgftransformshift { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
8361 \pgfnode { rectangle } { north~east }
8362 {
8363   \begin { minipage } { 20 cm }
8364   \raggedleft
8365   \@@_math_toggle: #6 \@@_math_toggle:
8366   \end { minipage }
8367 }
8368 { }
8369 { }

```

```

8370 \endpgfpicture
8371 }

```

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`.

```

8372 \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 `\`.

```

8373 \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`.

```

8374 \cs_new_protected:Npn \@@_CodeAfter_ii:n #1 \end
8375 {
8376   \tl_gput_right:Nn \g_nicematrix_code_after_tl { #1 }
8377   \@@_CodeAfter_iv:n
8378 }

```

We catch the argument of the command `\end` (in `#1`).

```

8379 \cs_new_protected:Npn \@@_CodeAfter_iv:n #1
8380 {

```

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.

```

8381   \str_if_eq:eeTF \@currenvir { #1 }
8382   { \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`.

```

8383   {
8384     \tl_gput_right:Nn \g_nicematrix_code_after_tl { \end { #1 } }
8385     \@@_CodeAfter_ii:n
8386   }
8387 }

```

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.

```

8388 \cs_new_protected:Npn \l_@@_delimiter:nnn #1 #2 #3
8389 {
8390   \pgfpicture
8391   \pgfrememberpicturepositiononpagetrue
8392   \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.

```

8393   \l_@@_qpoint:n { row - 1 }
8394   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
8395   \l_@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
8396   \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).

```

8397   \bool_if:nTF { #3 }
8398   { \dim_set_eq:NN \l_tmpa_dim \c_max_dim }
8399   { \dim_set:Nn \l_tmpa_dim { - \c_max_dim } }
8400   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
8401   {
8402     \cs_if_exist:cT
8403     { pgf @ sh @ ns @ \l_@@_env: - ##1 - #2 }
8404     {
8405       \pgfpointanchor
8406       { \l_@@_env: - ##1 - #2 }
8407       { \bool_if:nTF { #3 } { west } { east } }
8408       \dim_set:Nn \l_tmpa_dim
8409       { \bool_if:nTF { #3 } \dim_min:nn \dim_max:nn \l_tmpa_dim \pgf@x }
8410     }
8411   }

```

Now we can put the delimiter with a node of PGF.

```

8412   \pgfset { inner~sep = \c_zero_dim }
8413   \dim_zero:N \nulldelimiterspace
8414   \pgftransformshift
8415   {
8416     \pgfpoint
8417     { \l_tmpa_dim }
8418     { ( \l_@@_y_initial_dim + \l_@@_y_final_dim + \arrayrulewidth ) / 2 }
8419   }
8420   \pgfnode
8421   { rectangle }
8422   { \bool_if:nTF { #3 } { east } { west } }
8423   {

```

Here is the content of the PGF node, that is to say the delimiter, constructed with its right size.

```

8424     \nullfont
8425     \c_math_toggle_token
8426     \l_@@_color:o \l_@@_delimiters_color_tl
8427     \bool_if:nTF { #3 } { \left #1 } { \left . }
8428     \vcenter
8429     {
8430       \nullfont
8431       \hrule \@height
8432       \dim_eval:n { \l_@@_y_initial_dim - \l_@@_y_final_dim }
8433       \@depth \c_zero_dim
8434       \@width \c_zero_dim
8435     }
8436     \bool_if:nTF { #3 } { \right . } { \right #1 }
8437     \c_math_toggle_token
8438   }
8439   { }
8440   { }
8441   \endpgfpicture
8442 }

```


34 The command \SubMatrix

```

8443 \keys_define:nn { NiceMatrix / sub-matrix }
8444 {
8445   extra-height .dim_set:N = \l_@@_submatrix_extra_height_dim ,
8446   extra-height .value_required:n = true ,
8447   left-xshift .dim_set:N = \l_@@_submatrix_left_xshift_dim ,
8448   left-xshift .value_required:n = true ,
8449   right-xshift .dim_set:N = \l_@@_submatrix_right_xshift_dim ,
8450   right-xshift .value_required:n = true ,
8451   xshift .meta:n = { left-xshift = #1, right-xshift = #1 } ,
8452   xshift .value_required:n = true ,
8453   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8454   delimiters / color .value_required:n = true ,
8455   slim .bool_set:N = \l_@@_submatrix_slim_bool ,
8456   slim .default:n = true ,
8457   hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
8458   hlines .default:n = all ,
8459   vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
8460   vlines .default:n = all ,
8461   hvlines .meta:n = { hlines, vlines } ,
8462   hvlines .value_forbidden:n = true
8463 }
8464 \keys_define:nn { NiceMatrix }
8465 {
8466   SubMatrix .inherit:n = NiceMatrix / sub-matrix ,
8467   NiceArray / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
8468   pNiceArray / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
8469   NiceMatrixOptions / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
8470 }

```

The following keys set is for the command \SubMatrix itself (not the tuning of \SubMatrix that can be done elsewhere).

```

8471 \keys_define:nn { NiceMatrix / SubMatrix }
8472 {
8473   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8474   delimiters / color .value_required:n = true ,
8475   hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
8476   hlines .default:n = all ,
8477   vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
8478   vlines .default:n = all ,
8479   hvlines .meta:n = { hlines, vlines } ,
8480   hvlines .value_forbidden:n = true ,
8481   name .code:n =
8482     \tl_if_empty:nTF { #1 }
8483     { \@@_error:n { Invalid-name } }
8484     {
8485       \regex_match:nnTF { \A[A-Za-z][A-Za-z0-9]*\Z } { #1 }
8486       {
8487         \seq_if_in:NnTF \g_@@_submatrix_names_seq { #1 }
8488         { \@@_error:nn { Duplicate-name-for-SubMatrix } { #1 } }
8489         {
8490           \str_set:Nn \l_@@_submatrix_name_str { #1 }
8491           \seq_gput_right:Nn \g_@@_submatrix_names_seq { #1 }
8492         }
8493       }
8494       { \@@_error:n { Invalid-name } }
8495     } ,
8496   name .value_required:n = true ,
8497   rules .code:n = \keys_set:nn { NiceMatrix / rules } { #1 } ,
8498   rules .value_required:n = true ,
8499   code .tl_set:N = \l_@@_code_tl ,

```

```

8500     code .value_required:n = true ,
8501     unknown .code:n = \@_error:n { Unknown~key~for~SubMatrix }
8502 }

8503 \NewDocumentCommand \@_SubMatrix_in_code_before { m m m m ! 0 { } }
8504 {
8505     \peek_remove_spaces:n
8506     {
8507         \tl_gput_right:Nx \g_@@_pre_code_after_tl
8508         {
8509             \SubMatrix { #1 } { #2 } { #3 } { #4 }
8510             [
8511                 delimiters / color = \l_@@_delimiters_color_tl ,
8512                 hlines = \l_@@_submatrix_hlines_clist ,
8513                 vlines = \l_@@_submatrix_vlines_clist ,
8514                 extra-height = \dim_use:N \l_@@_submatrix_extra_height_dim ,
8515                 left-xshift = \dim_use:N \l_@@_submatrix_left_xshift_dim ,
8516                 right-xshift = \dim_use:N \l_@@_submatrix_right_xshift_dim ,
8517                 slim = \bool_to_str:N \l_@@_submatrix_slim_bool ,
8518                 #5
8519             ]
8520         }
8521         \@_SubMatrix_in_code_before_i { #2 } { #3 }
8522     }
8523 }

8524 \NewDocumentCommand \@_SubMatrix_in_code_before_i
8525 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
8526 { \@_SubMatrix_in_code_before_i:nnnn #1 #2 }

8527 \cs_new_protected:Npn \@_SubMatrix_in_code_before_i:nnnn #1 #2 #3 #4
8528 {
8529     \seq_gput_right:Nx \g_@@_submatrix_seq
8530     {
We use \str_if_eq:nnTF because it is fully expandable.
8531         { \str_if_eq:nnTF { #1 } { last } { \int_use:N \c@iRow } { #1 } }
8532         { \str_if_eq:nnTF { #2 } { last } { \int_use:N \c@jCol } { #2 } }
8533         { \str_if_eq:nnTF { #3 } { last } { \int_use:N \c@iRow } { #3 } }
8534         { \str_if_eq:nnTF { #4 } { last } { \int_use:N \c@jCol } { #4 } }
8535     }
8536 }

```

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.

```

8537 \hook_gput_code:nnn { begindocument } { . }
8538 {
8539     \cs_set_nopar:Npn \l_@@_argspec_tl { m m m m 0 { } E { _ ^ } { { } { } } }
8540     \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl

```

```

8541 \exp_args:NNo \NewDocumentCommand \@@_SubMatrix \l_@@_argspec_tl
8542 {
8543   \peek_remove_spaces:n
8544   {
8545     \@@_sub_matrix:nnnnnnn
8546     { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } { #7 }
8547   }
8548 }
8549 }

```

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).

```

8550 \NewDocumentCommand \@@_compute_i_j:nn
8551 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
8552 { \@@_compute_i_j:nnnn #1 #2 }

8553 \cs_new_protected:Npn \@@_compute_i_j:nnnn #1 #2 #3 #4
8554 {
8555   \cs_set_nopar:Npn \l_@@_first_i_tl { #1 }
8556   \cs_set_nopar:Npn \l_@@_first_j_tl { #2 }
8557   \cs_set_nopar:Npn \l_@@_last_i_tl { #3 }
8558   \cs_set_nopar:Npn \l_@@_last_j_tl { #4 }
8559   \tl_if_eq:NnT \l_@@_first_i_tl { last }
8560     { \tl_set:NV \l_@@_first_i_tl \c@iRow }
8561   \tl_if_eq:NnT \l_@@_first_j_tl { last }
8562     { \tl_set:NV \l_@@_first_j_tl \c@jCol }
8563   \tl_if_eq:NnT \l_@@_last_i_tl { last }
8564     { \tl_set:NV \l_@@_last_i_tl \c@iRow }
8565   \tl_if_eq:NnT \l_@@_last_j_tl { last }
8566     { \tl_set:NV \l_@@_last_j_tl \c@jCol }
8567 }

8568 \cs_new_protected:Npn \@@_sub_matrix:nnnnnnn #1 #2 #3 #4 #5 #6 #7
8569 {
8570   \group_begin:

```

The four following token lists correspond to the position of the `\SubMatrix`.

```

8571 \@@_compute_i_j:nn { #2 } { #3 }
8572 \int_compare:NnT \l_@@_first_i_tl = \l_@@_last_i_tl
8573   { \cs_set_nopar:Npn \arraystretch { 1 } }
8574 \bool_lazy_or:nnTF
8575   { \int_compare_p:Nn \l_@@_last_i_tl > \g_@@_row_total_int }
8576   { \int_compare_p:Nn \l_@@_last_j_tl > \g_@@_col_total_int }
8577   { \@@_error:nn { Construct~too~large } { \SubMatrix } }
8578 {
8579   \str_clear_new:N \l_@@_submatrix_name_str
8580   \keys_set:nn { NiceMatrix / SubMatrix } { #5 }
8581   \pgfpicture
8582   \pgfrememberpicturepositiononpagetrue
8583   \pgf@relevantforpicturesizefalse
8584   \pgfset { inner~sep = \c_zero_dim }
8585   \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
8586   \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }

```

The last value of `\int_step_inline:nnn` is provided by curriification.

```

8587 \bool_if:NTF \l_@@_submatrix_slim_bool
8588   { \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl }
8589   { \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int }
8590   {
8591     \cs_if_exist:cT
8592       { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
8593       {
8594         \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
8595         \dim_set:Nn \l_@@_x_initial_dim

```

```

8596         { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
8597     }
8598     \cs_if_exist:cT
8599     { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
8600     {
8601         \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
8602         \dim_set:Nn \l_@@_x_final_dim
8603         { \dim_max:nn \l_@@_x_final_dim \pgf@x }
8604     }
8605 }
8606 \dim_compare:nNnTF \l_@@_x_initial_dim = \c_max_dim
8607 { \@@_error:nn { Impossible-delimiter } { left } }
8608 {
8609     \dim_compare:nNnTF \l_@@_x_final_dim = { - \c_max_dim }
8610     { \@@_error:nn { Impossible-delimiter } { right } }
8611     { \@@_sub_matrix_i:nnnn { #1 } { #4 } { #6 } { #7 } }
8612 }
8613 \endpgfpicture
8614 }
8615 \group_end:
8616 }

```

#1 is the left delimiter, #2 is the right one, #3 is the subscript and #4 is the superscript.

```

8617 \cs_new_protected:Npn \@@_sub_matrix_i:nnnn #1 #2 #3 #4
8618 {
8619     \@@_qpoint:n { row - \l_@@_first_i_tl - base }
8620     \dim_set:Nn \l_@@_y_initial_dim
8621     {
8622         \fp_to_dim:n
8623         {
8624             \pgf@y
8625             + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
8626         }
8627     }
8628     \@@_qpoint:n { row - \l_@@_last_i_tl - base }
8629     \dim_set:Nn \l_@@_y_final_dim
8630     { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
8631     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
8632     {
8633         \cs_if_exist:cT
8634         { pgf @ sh @ ns @ \@@_env: - \l_@@_first_i_tl - ##1 }
8635         {
8636             \pgfpointanchor { \@@_env: - \l_@@_first_i_tl - ##1 } { north }
8637             \dim_set:Nn \l_@@_y_initial_dim
8638             { \dim_max:nn \l_@@_y_initial_dim \pgf@y }
8639         }
8640         \cs_if_exist:cT
8641         { pgf @ sh @ ns @ \@@_env: - \l_@@_last_i_tl - ##1 }
8642         {
8643             \pgfpointanchor { \@@_env: - \l_@@_last_i_tl - ##1 } { south }
8644             \dim_set:Nn \l_@@_y_final_dim
8645             { \dim_min:nn \l_@@_y_final_dim \pgf@y }
8646         }
8647     }
8648     \dim_set:Nn \l_tmpa_dim
8649     {
8650         \l_@@_y_initial_dim - \l_@@_y_final_dim +
8651         \l_@@_submatrix_extra_height_dim - \arrayrulewidth
8652     }
8653     \dim_zero:N \nulldelimiterspace

```

We will draw the rules in the `\SubMatrix`.

```

8654 \group_begin:
8655 \pgfsetlinewidth { 1.1 \arrayrulewidth }
8656 \@@_set_CT@arc@:o \l_@@_rules_color_tl
8657 \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`.

```

8658 \seq_map_inline:Nn \g_@@_cols_vlism_seq
8659 {
8660   \int_compare:nNnT \l_@@_first_j_tl < { ##1 }
8661   {
8662     \int_compare:nNnT
8663       { ##1 } < { \int_eval:n { \l_@@_last_j_tl + 1 } }
8664     {

```

First, we extract the value of the abscissa of the rule we have to draw.

```

8665       \@@_qpoint:n { col - ##1 }
8666       \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
8667       \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
8668       \pgfusepathqstroke
8669     }
8670   }
8671 }

```

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.

```

8672 \tl_if_eq:NNTF \l_@@_submatrix_vlines_clist \c_@@_all_tl
8673 { \int_step_inline:nn { \l_@@_last_j_tl - \l_@@_first_j_tl } }
8674 { \clist_map_inline:Nn \l_@@_submatrix_vlines_clist }
8675 {
8676   \bool_lazy_and:nnTF
8677     { \int_compare_p:nNn { ##1 } > \c_zero_int }
8678     {
8679       \int_compare_p:nNn
8680         { ##1 } < { \l_@@_last_j_tl - \l_@@_first_j_tl + 1 } }
8681       {
8682         \@@_qpoint:n { col - \int_eval:n { ##1 + \l_@@_first_j_tl } }
8683         \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
8684         \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
8685         \pgfusepathqstroke
8686       }
8687       { \@@_error:nnn { Wrong~line~in~SubMatrix } { vertical } { ##1 } }
8688     }

```

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.

```

8689 \tl_if_eq:NNTF \l_@@_submatrix_hlines_clist \c_@@_all_tl
8690 { \int_step_inline:nn { \l_@@_last_i_tl - \l_@@_first_i_tl } }
8691 { \clist_map_inline:Nn \l_@@_submatrix_hlines_clist }
8692 {
8693   \bool_lazy_and:nnTF
8694     { \int_compare_p:nNn { ##1 } > \c_zero_int }
8695     {
8696       \int_compare_p:nNn
8697         { ##1 } < { \l_@@_last_i_tl - \l_@@_first_i_tl + 1 } }
8698       {
8699         \@@_qpoint:n { row - \int_eval:n { ##1 + \l_@@_first_i_tl } }

```

We use a group to protect `\l_tmpa_dim` and `\l_tmpb_dim`.

```

8700 \group_begin:

```

We compute in `\l_tmpa_dim` the x -value of the left end of the rule.

```

8701 \dim_set:Nn \l_tmpa_dim

```

```

8702         { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
8703     \str_case:nn { #1 }
8704     {
8705         ( { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
8706         [ { \dim_sub:Nn \l_tmpa_dim { 0.2 mm } }
8707         \{ { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
8708         }
8709     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }

```

We compute in `\l_tmpb_dim` the x -value of the right end of the rule.

```

8710     \dim_set:Nn \l_tmpb_dim
8711     { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
8712     \str_case:nn { #2 }
8713     {
8714         ) { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
8715         ] { \dim_add:Nn \l_tmpb_dim { 0.2 mm } }
8716         \} { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
8717     }
8718     \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
8719     \pgfusepathqstroke
8720     \group_end:
8721 }
8722 { \@@_error:nnn { Wrong~line~in~SubMatrix } { horizontal } { ##1 } }
8723 }

```

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).

```

8724     \str_if_empty:NF \l_@@_submatrix_name_str
8725     {
8726         \@@_pgf_rect_node:nnnnn \l_@@_submatrix_name_str
8727         \l_@@_x_initial_dim \l_@@_y_initial_dim
8728         \l_@@_x_final_dim \l_@@_y_final_dim
8729     }
8730     \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`.

```

8731     \begin { pgfscope }
8732     \pgftransformshift
8733     {
8734         \pgfpoint
8735         { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
8736         { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
8737     }
8738     \str_if_empty:NTF \l_@@_submatrix_name_str
8739     { \@@_node_left:nn #1 { } }
8740     { \@@_node_left:nn #1 { \@@_env: - \l_@@_submatrix_name_str - left } }
8741     \end { pgfscope }

```

Now, we deal with the right delimiter.

```

8742     \pgftransformshift
8743     {
8744         \pgfpoint
8745         { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
8746         { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
8747     }
8748     \str_if_empty:NTF \l_@@_submatrix_name_str
8749     { \@@_node_right:nnnn #2 { } { #3 } { #4 } }
8750     {
8751         \@@_node_right:nnnn #2
8752         { \@@_env: - \l_@@_submatrix_name_str - right } { #3 } { #4 }
8753     }

```

```

8754 \cs_set_eq:NN \pgfpointanchor \@@_pgfpointanchor:n
8755 \flag_clear_new:n { nicematrix }
8756 \l_@@_code_tl
8757 }

```

In the key code of the command `\SubMatrix` there may be Tikz instructions. We want that, in these instructions, the i and j in specifications of nodes of the forms $i-j$, $\text{row-}i$, $\text{col-}j$ and $i-|j$ refer to the number of row and column *relative* of the current `\SubMatrix`. That's why we will patch (locally in the `\SubMatrix`) the command `\pgfpointanchor`.

```

8758 \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.

```

8759 \cs_new_protected:Npn \@@_pgfpointanchor:n #1
8760 {
8761   \use:e
8762   { \exp_not:N \@@_old_pgfpointanchor { \@@_pgfpointanchor_i:nn #1 } }
8763 }

```

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 -.

```

8764 \cs_new:Npn \@@_pgfpointanchor_i:nn #1 #2
8765 { #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.

```

8766 \tl_const:Nn \c_@@_integers_alist_tl
8767 {
8768   { 1 } { } { 2 } { } { 3 } { } { 4 } { } { 5 } { }
8769   { 6 } { } { 7 } { } { 8 } { } { 9 } { } { 10 } { }
8770   { 11 } { } { 12 } { } { 13 } { } { 14 } { } { 15 } { }
8771   { 16 } { } { 17 } { } { 18 } { } { 19 } { } { 20 } { }
8772 }

```

```

8773 \cs_new:Npn \@@_pgfpointanchor_ii:w #1-#2\q_stop
8774 {

```

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`.

```

8775 \tl_if_empty:nTF { #2 }
8776 {
8777   \str_case:nVTF { #1 } \c_@@_integers_alist_tl
8778   {
8779     \flag_raise:n { nicematrix }
8780     \int_if_even:nTF { \flag_height:n { nicematrix } }
8781     { \int_eval:n { #1 + \l_@@_first_i_tl - 1 } }
8782     { \int_eval:n { #1 + \l_@@_first_j_tl - 1 } }
8783   }
8784   { #1 }
8785 }

```

If there is an hyphen, we have to see whether we have a node of the form $i-j$, $\text{row-}i$ or $\text{col-}j$.

```

8786     { \@@_pgfpointanchor_iii:w { #1 } #2 }
8787 }

```

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`).

```

8788 \cs_new:Npn \@@_pgfpointanchor_iii:w #1 #2 -
8789 {
8790   \str_case:nnF { #1 }
8791   {
8792     { row } { row - \int_eval:n { #2 + \l_@@_first_i_tl - 1 } }
8793     { col } { col - \int_eval:n { #2 + \l_@@_first_j_tl - 1 } }
8794   }

```

Now the case of a node of the form $i-j$.

```

8795   {
8796     \int_eval:n { #1 + \l_@@_first_i_tl - 1 }
8797     - \int_eval:n { #2 + \l_@@_first_j_tl - 1 }
8798   }
8799 }

```

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`).

```

8800 \cs_new_protected:Npn \@@_node_left:nn #1 #2
8801 {
8802   \pgfnode
8803   { rectangle }
8804   { east }
8805   {
8806     \nullfont
8807     \c_math_toggle_token
8808     \@@_color:o \l_@@_delimiters_color_tl
8809     \left #1
8810     \vcenter
8811     {
8812       \nullfont
8813       \hrule \@height \l_tmpa_dim
8814       \c_zero_dim
8815       \width \c_zero_dim
8816     }
8817     \right .
8818     \c_math_toggle_token
8819   }
8820   { #2 }
8821   { }
8822 }

```

The command `\@@_node_right:nn` puts the right delimiter with the correct size. The argument `#1` is the delimiter to put. The argument `#2` is the name we will give to this PGF node (if the key `name` has been used in `\SubMatrix`). The argument `#3` is the subscript and `#4` is the superscript.

```

8823 \cs_new_protected:Npn \@@_node_right:nnnn #1 #2 #3 #4
8824 {
8825   \pgfnode
8826   { rectangle }
8827   { west }
8828   {
8829     \nullfont
8830     \c_math_toggle_token
8831     \@@_color:o \l_@@_delimiters_color_tl
8832     \left .
8833     \vcenter

```



```

8834     {
8835         \nullfont
8836         \hrule \@height \l_tmpa_dim
8837             \@depth \c_zero_dim
8838             \@width \c_zero_dim
8839     }
8840     \right #1
8841     \tl_if_empty:nF { #3 } { _ { \smash { #3 } } }
8842     ^ { \smash { #4 } }
8843     \c_math_toggle_token
8844 }
8845 { #2 }
8846 { }
8847 }

```

35 Les commandes \UnderBrace et \OverBrace

The following commands will be linked to \UnderBrace and \OverBrace in the \CodeAfter.

```

8848 \NewDocumentCommand \@@_UnderBrace { 0 { } m m m 0 { } }
8849 {
8850     \peek_remove_spaces:n
8851     { \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { under } }
8852 }
8853 \NewDocumentCommand \@@_OverBrace { 0 { } m m m 0 { } }
8854 {
8855     \peek_remove_spaces:n
8856     { \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { over } }
8857 }
8858 \keys_define:nn { NiceMatrix / Brace }
8859 {
8860     left-shorten .bool_set:N = \l_@@_brace_left_shorten_bool ,
8861     left-shorten .default:n = true ,
8862     right-shorten .bool_set:N = \l_@@_brace_right_shorten_bool ,
8863     shorten .meta:n = { left-shorten , right-shorten } ,
8864     right-shorten .default:n = true ,
8865     yshift .dim_set:N = \l_@@_brace_yshift_dim ,
8866     yshift .value_required:n = true ,
8867     yshift .initial:n = \c_zero_dim ,
8868     color .tl_set:N = \l_tmpa_tl ,
8869     color .value_required:n = true ,
8870     unknown .code:n = \@@_error:n { Unknown~key~for~Brace }
8871 }

```

#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*.

```

8872 \cs_new_protected:Npn \@@_brace:nnnnn #1 #2 #3 #4 #5
8873 {
8874     \group_begin:

```

The four following token lists correspond to the position of the sub-matrix to which a brace will be attached.

```

8875 \@@_compute_i_j:nn { #1 } { #2 }
8876 \bool_lazy_or:nnTF
8877 { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
8878 { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
8879 {
8880     \str_if_eq:nnTF { #5 } { under }

```

```

8881 { \@@_error:nn { Construct-too-large } { \UnderBrace } }
8882 { \@@_error:nn { Construct-too-large } { \OverBrace } }
8883 }
8884 {
8885 \tl_clear:N \l_tmpa_tl
8886 \keys_set:nn { NiceMatrix / Brace } { #4 }
8887 \tl_if_empty:NF \l_tmpa_tl { \color { \l_tmpa_tl } }
8888 \pgfpicture
8889 \pgfrememberpicturepositiononpagetrue
8890 \pgf@relevantforpicturesizefalse
8891 \bool_if:NT \l_@@_brace_left_shorten_bool
8892 {
8893 \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
8894 \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
8895 {
8896 \cs_if_exist:cT
8897 { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
8898 {
8899 \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
8900 \dim_set:Nn \l_@@_x_initial_dim
8901 { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
8902 }
8903 }
8904 }
8905 \bool_lazy_or:nnT
8906 { \bool_not_p:n \l_@@_brace_left_shorten_bool }
8907 { \dim_compare_p:nNn \l_@@_x_initial_dim = \c_max_dim }
8908 {
8909 \@@_qpoint:n { col - \l_@@_first_j_tl }
8910 \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
8911 }
8912 \bool_if:NT \l_@@_brace_right_shorten_bool
8913 {
8914 \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
8915 \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
8916 {
8917 \cs_if_exist:cT
8918 { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
8919 {
8920 \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
8921 \dim_set:Nn \l_@@_x_final_dim
8922 { \dim_max:nn \l_@@_x_final_dim \pgf@x }
8923 }
8924 }
8925 }
8926 \bool_lazy_or:nnT
8927 { \bool_not_p:n \l_@@_brace_right_shorten_bool }
8928 { \dim_compare_p:nNn \l_@@_x_final_dim = { - \c_max_dim } }
8929 {
8930 \@@_qpoint:n { col - \int_eval:n { \l_@@_last_j_tl + 1 } }
8931 \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
8932 }
8933 \pgfset { inner~sep = \c_zero_dim }
8934 \str_if_eq:nnTF { #5 } { under }
8935 { \@@_underbrace_i:n { #3 } }
8936 { \@@_overbrace_i:n { #3 } }
8937 \endpgfpicture
8938 }
8939 \group_end:
8940 }

```

The argument is the text to put above the brace.

```

8941 \cs_new_protected:Npn \@@_overbrace_i:n #1
8942 {

```

```

8943 \@@_qpoint:n { row - \l_@@_first_i_tl }
8944 \pgftransformshift
8945 {
8946   \pgfpoint
8947   { ( \l_@@_x_initial_dim + \l_@@_x_final_dim) / 2 }
8948   { \pgf@y + \l_@@_brace_yshift_dim - 3 pt}
8949 }
8950 \pgfnode
8951 { rectangle }
8952 { south }
8953 {
8954   \vtop
8955   {
8956     \group_begin:
8957     \everycr { }
8958     \halign
8959     {
8960       \hfil ## \hfil \crcr
8961       \@@_math_toggle: #1 \@@_math_toggle: \cr
8962       \noalign { \skip_vertical:n { 3 pt } \nointerlineskip }
8963       \c_math_toggle_token
8964       \overbrace
8965       {
8966         \hbox_to_wd:nn
8967         { \l_@@_x_final_dim - \l_@@_x_initial_dim }
8968         { }
8969       }
8970       \c_math_toggle_token
8971       \cr
8972       }
8973     \group_end:
8974   }
8975 }
8976 { }
8977 { }
8978 }

```

The argument is the text to put under the brace.

```

8979 \cs_new_protected:Npn \@@_underbrace_i:n #1
8980 {
8981   \@@_qpoint:n { row - \int_eval:n { \l_@@_last_i_tl + 1 } }
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   { north }
8991   {
8992     \group_begin:
8993     \everycr { }
8994     \vbox
8995     {
8996       \halign
8997       {
8998         \hfil ## \hfil \crcr
8999         \c_math_toggle_token
9000         \underbrace
9001         {
9002           \hbox_to_wd:nn
9003           { \l_@@_x_final_dim - \l_@@_x_initial_dim }
9004           { }

```

```

9005         }
9006         \c_math_toggle_token
9007         \cr
9008         \noalign { \skip_vertical:n { 3 pt } \nointerlineskip }
9009         \@@_math_toggle: #1 \@@_math_toggle: \cr
9010     }
9011 }
9012 \group_end:
9013 }
9014 { }
9015 { }
9016 }

```

36 The command TikzEveryCell

```

9017 \bool_new:N \l_@@_not_empty_bool
9018 \bool_new:N \l_@@_empty_bool
9019
9020 \keys_define:nn { NiceMatrix / TikzEveryCell }
9021 {
9022     not-empty .code:n =
9023         \bool_lazy_or:nnTF
9024             \l_@@_in_code_after_bool
9025             \g_@@_recreate_cell_nodes_bool
9026             { \bool_set_true:N \l_@@_not_empty_bool }
9027             { \@@_error:n { detection-of-empty-cells } } ,
9028     not-empty .value_forbidden:n = true ,
9029     empty .code:n =
9030         \bool_lazy_or:nnTF
9031             \l_@@_in_code_after_bool
9032             \g_@@_recreate_cell_nodes_bool
9033             { \bool_set_true:N \l_@@_empty_bool }
9034             { \@@_error:n { detection-of-empty-cells } } ,
9035     empty .value_forbidden:n = true ,
9036     unknown .code:n = \@@_error:n { Unknown-key-for-TikzEveryCell }
9037 }
9038
9039
9040 \NewDocumentCommand { \@@_TikzEveryCell } { 0 { } m }
9041 {
9042     \IfPackageLoadedTF { tikz }
9043     {
9044         \group_begin:
9045         \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.

```

9046         \tl_set:Nn \l_tmpa_tl { { #2 } }
9047         \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
9048             { \@@_for_a_block:nnnnn #1 }
9049         \@@_all_the_cells:
9050         \group_end:
9051     }
9052     { \@@_error:n { TikzEveryCell-without-tikz } }
9053 }
9054
9055 \tl_new:N \@@_i_tl
9056 \tl_new:N \@@_j_tl
9057
9058 \cs_new_protected:Nn \@@_all_the_cells:

```

```

9059 {
9060   \int_step_variable:nNn { \int_use:c { c@iRow } } \@@_i_tl
9061   {
9062     \int_step_variable:nNn { \int_use:c { c@jCol } } \@@_j_tl
9063     {
9064       \cs_if_exist:cF { cell - \@@_i_tl - \@@_j_tl }
9065       {
9066         \exp_args:NNe \seq_if_in:NnF \l_@@_corners_cells_seq
9067         { \@@_i_tl - \@@_j_tl }
9068         {
9069           \bool_set_false:N \l_tmpa_bool
9070           \cs_if_exist:cTF
9071           { pgf @ sh @ ns @ \@@_env: - \@@_i_tl - \@@_j_tl }
9072           {
9073             \bool_if:NF \l_@@_empty_bool
9074             { \bool_set_true:N \l_tmpa_bool }
9075           }
9076           {
9077             \bool_if:NF \l_@@_not_empty_bool
9078             { \bool_set_true:N \l_tmpa_bool }
9079           }
9080           \bool_if:NT \l_tmpa_bool
9081           {
9082             \@@_block_tikz:nnnnV
9083             \@@_i_tl \@@_j_tl \@@_i_tl \@@_j_tl \l_tmpa_tl
9084           }
9085         }
9086       }
9087     }
9088   }
9089 }
9090
9091 \cs_new_protected:Nn \@@_for_a_block:nnnnn
9092 {
9093   \bool_if:NF \l_@@_empty_bool
9094   {
9095     \@@_block_tikz:nnnnV
9096     { #1 } { #2 } { #3 } { #4 } \l_tmpa_tl
9097   }
9098   \@@_mark_cells_of_block:nnnn { #1 } { #2 } { #3 } { #4 }
9099 }
9100
9101 \cs_new_protected:Nn \@@_mark_cells_of_block:nnnn
9102 {
9103   \int_step_inline:nnn { #1 } { #3 }
9104   {
9105     \int_step_inline:nnn { #2 } { #4 }
9106     { \cs_set:cpn { cell - ##1 - ####1 } { } }
9107   }
9108 }

```

37 The command \ShowCellNames

```

9109 \NewDocumentCommand \@@_ShowCellNames_CodeBefore { }
9110 {
9111   \dim_gzero_new:N \g_@@_tmpc_dim
9112   \dim_gzero_new:N \g_@@_tmpd_dim
9113   \dim_gzero_new:N \g_@@_tmpe_dim
9114   \int_step_inline:nn \c@iRow
9115   {
9116     \begin { pgfpicture }
9117     \@@_qpoint:n { row - ##1 }

```

```

9118 \dim_set_eq:NN \l_tmpa_dim \pgf@y
9119 \@@_qpoint:n { row - \int_eval:n { ##1 + 1 } }
9120 \dim_gset:Nn \g_tmpa_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
9121 \dim_gset:Nn \g_tmpb_dim { \l_tmpa_dim - \pgf@y }
9122 \bool_if:NTF \l_@@_in_code_after_bool
9123 \end { pgfpicture }
9124 \int_step_inline:nn \c@jCol
9125 {
9126   \hbox_set:Nn \l_tmpa_box
9127     { \normalfont \Large \color { red ! 50 } ##1 - ####1 }
9128   \begin { pgfpicture }
9129     \@@_qpoint:n { col - ####1 }
9130     \dim_gset_eq:NN \g_@@_tmpc_dim \pgf@x
9131     \@@_qpoint:n { col - \int_eval:n { ####1 + 1 } }
9132     \dim_gset:Nn \g_@@_tmpd_dim { \pgf@x - \g_@@_tmpc_dim }
9133     \dim_gset_eq:NN \g_@@_tmpe_dim \pgf@x
9134     \endpgfpicture
9135   \end { pgfpicture }
9136   \fp_set:Nn \l_tmpa_fp
9137     {
9138       \fp_min:nn
9139       {
9140         \fp_min:nn
9141         {
9142           \dim_ratio:nn
9143             { \g_@@_tmpd_dim }
9144             { \box_wd:N \l_tmpa_box }
9145         }
9146         {
9147           \dim_ratio:nn
9148             { \g_tmpb_dim }
9149             { \box_ht_plus_dp:N \l_tmpa_box }
9150         }
9151       }
9152       { 1.0 }
9153     }
9154   \box_scale:Nnn \l_tmpa_box
9155     { \fp_use:N \l_tmpa_fp }
9156     { \fp_use:N \l_tmpa_fp }
9157   \pgfpicture
9158   \pgfrememberpicturepositiononpagetrue
9159   \pgf@relevantforpicturesizefalse
9160   \pgftransformshift
9161     {
9162       \pgfpoint
9163         { 0.5 * ( \g_@@_tmpc_dim + \g_@@_tmpe_dim ) }
9164         { \dim_use:N \g_tmpa_dim }
9165     }
9166   \pgfnode
9167     { rectangle }
9168     { center }
9169     { \box_use:N \l_tmpa_box }
9170     { }
9171     { }
9172   \endpgfpicture
9173 }
9174 }
9175 }
9176 \NewDocumentCommand \@@_ShowCellNames { }
9177 {
9178   \bool_if:NT \l_@@_in_code_after_bool
9179   {
9180     \pgfpicture

```

```

9181 \pgfrememberpicturepositiononpagetrue
9182 \pgf@relevantforpicturesizefalse
9183 \pgfpathrectanglecorners
9184 { \@@_qpoint:n { 1 } }
9185 {
9186 \@@_qpoint:n
9187 { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
9188 }
9189 \pgfsetfillopacity { 0.75 }
9190 \pgfsetfillcolor { white }
9191 \pgfusepathqfill
9192 \endpgfpicture
9193 }
9194 \dim_gzero_new:N \g_@@_tmpc_dim
9195 \dim_gzero_new:N \g_@@_tmpd_dim
9196 \dim_gzero_new:N \g_@@_tmpe_dim
9197 \int_step_inline:nn \c@iRow
9198 {
9199 \bool_if:NTF \l_@@_in_code_after_bool
9200 {
9201 \pgfpicture
9202 \pgfrememberpicturepositiononpagetrue
9203 \pgf@relevantforpicturesizefalse
9204 }
9205 { \begin { pgfpicture } }
9206 \@@_qpoint:n { row - ##1 }
9207 \dim_set_eq:NN \l_tmpa_dim \pgf@y
9208 \@@_qpoint:n { row - \int_eval:n { ##1 + 1 } }
9209 \dim_gset:Nn \g_tmpa_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
9210 \dim_gset:Nn \g_tmpb_dim { \l_tmpa_dim - \pgf@y }
9211 \bool_if:NTF \l_@@_in_code_after_bool
9212 { \endpgfpicture }
9213 { \end { pgfpicture } }
9214 \int_step_inline:nn \c@jCol
9215 {
9216 \hbox_set:Nn \l_tmpa_box
9217 {
9218 \normalfont \Large \sffamily \bfseries
9219 \bool_if:NTF \l_@@_in_code_after_bool
9220 { \color { red } }
9221 { \color { red ! 50 } }
9222 ##1 - ####1
9223 }
9224 \bool_if:NTF \l_@@_in_code_after_bool
9225 {
9226 \pgfpicture
9227 \pgfrememberpicturepositiononpagetrue
9228 \pgf@relevantforpicturesizefalse
9229 }
9230 { \begin { pgfpicture } }
9231 \@@_qpoint:n { col - ####1 }
9232 \dim_gset_eq:NN \g_@@_tmpc_dim \pgf@x
9233 \@@_qpoint:n { col - \int_eval:n { ####1 + 1 } }
9234 \dim_gset:Nn \g_@@_tmpd_dim { \pgf@x - \g_@@_tmpc_dim }
9235 \dim_gset_eq:NN \g_@@_tmpe_dim \pgf@x
9236 \bool_if:NTF \l_@@_in_code_after_bool
9237 { \endpgfpicture }
9238 { \end { pgfpicture } }
9239 \fp_set:Nn \l_tmpa_fp
9240 {
9241 \fp_min:nn
9242 {
9243 \fp_min:nn

```

```

9244         { \dim_ratio:nn \g_@@_tmpd_dim { \box_wd:N \l_tmpa_box } }
9245         { \dim_ratio:nn \g_tmpb_dim { \box_ht_plus_dp:N \l_tmpa_box } }
9246     }
9247     { 1.0 }
9248 }
9249 \box_scale:Nnn \l_tmpa_box { \fp_use:N \l_tmpa_fp } { \fp_use:N \l_tmpa_fp }
9250 \pgfpicture
9251 \pgfrememberpicturepositiononpagetrue
9252 \pgf@relevantforpicturesizefalse
9253 \pgftransformshift
9254 {
9255     \pgfpoint
9256     { 0.5 * ( \g_@@_tmpc_dim + \g_@@_tmpe_dim ) }
9257     { \dim_use:N \g_tmpa_dim }
9258 }
9259 \pgfnode
9260 { rectangle }
9261 { center }
9262 { \box_use:N \l_tmpa_box }
9263 { }
9264 { }
9265 \endpgfpicture
9266 }
9267 }
9268 }

```

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.

```

9269 \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.

```

9270 \bool_new:N \g_@@_footnote_bool

9271 \msg_new:nnnn { nicematrix } { Unknown~key~for~package }
9272 {
9273     The~key~'\l_keys_key_str'~is~unknown. \\
9274     That~key~will~be~ignored. \\
9275     For~a~list~of~the~available~keys,~type~H~<return>.
9276 }
9277 {
9278     The~available~keys~are~(in~alphabetic~order):~
9279     footnote,~
9280     footnotehyper,~
9281     messages-for-Overleaf,~
9282     no-test-for-array,~
9283     renew-dots,~and~
9284     renew-matrix.
9285 }

9286 \keys_define:nn { NiceMatrix / Package }
9287 {
9288     renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
9289     renew-dots .value_forbidden:n = true ,
9290     renew-matrix .code:n = \@@_renew_matrix: ,

```



```

9291     renew-matrix .value_forbidden:n = true ,
9292     messages-for-Overleaf .bool_set:N = \g_@@_messages_for_Overleaf_bool ,
9293     footnote .bool_set:N = \g_@@_footnote_bool ,
9294     footnotehyper .bool_set:N = \g_@@_footnotehyper_bool ,
9295     no-test-for-array .bool_set:N = \g_@@_no_test_for_array_bool ,
9296     no-test-for-array .default:n = true ,
9297     unknown .code:n = \@@_error:n { Unknown-key-for-package }
9298   }
9299 \ProcessKeysOptions { NiceMatrix / Package }

9300 \@@_msg_new:nn { footnote-with-footnotehyper-package }
9301 {
9302   You-can't-use-the-option-'footnote'~because-the-package~
9303   footnotehyper-has-already-been-loaded.~
9304   If-you-want,-you-can-use-the-option-'footnotehyper'~and-the-footnotes~
9305   within-the-environments-of-nicematrix-will-be-extracted-with-the-tools~
9306   of-the-package-footnotehyper.\\
9307   The-package-footnote-won't-be-loaded.
9308 }

9309 \@@_msg_new:nn { footnotehyper-with-footnote-package }
9310 {
9311   You-can't-use-the-option-'footnotehyper'~because-the-package~
9312   footnote-has-already-been-loaded.~
9313   If-you-want,-you-can-use-the-option-'footnote'~and-the-footnotes~
9314   within-the-environments-of-nicematrix-will-be-extracted-with-the-tools~
9315   of-the-package-footnote.\\
9316   The-package-footnotehyper-won't-be-loaded.
9317 }

9318 \bool_if:NT \g_@@_footnote_bool
9319 {

```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```

9320   \IfClassLoadedTF { beamer }
9321   { \bool_set_false:N \g_@@_footnote_bool }
9322   {
9323     \IfPackageLoadedTF { footnotehyper }
9324     { \@@_error:n { footnote-with-footnotehyper-package } }
9325     { \usepackage { footnote } }
9326   }
9327 }

9328 \bool_if:NT \g_@@_footnotehyper_bool
9329 {

```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```

9330   \IfClassLoadedTF { beamer }
9331   { \bool_set_false:N \g_@@_footnote_bool }
9332   {
9333     \IfPackageLoadedTF { footnote }
9334     { \@@_error:n { footnotehyper-with-footnote-package } }
9335     { \usepackage { footnotehyper } }
9336   }
9337   \bool_set_true:N \g_@@_footnote_bool
9338 }

```

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.

```
9339 \bool_new:N \l_@@_underscore_loaded_bool
9340 \IfPackageLoadedTF { underscore }
9341 { \bool_set_true:N \l_@@_underscore_loaded_bool }
9342 { }
9343 \hook_gput_code:nnn { begindocument } { . }
9344 {
9345   \bool_if:NF \l_@@_underscore_loaded_bool
9346   {
9347     \IfPackageLoadedTF { underscore }
9348     { \@@_error:n { underscore~after~nicematrix } }
9349     { }
9350   }
9351 }
```

40 Error messages of the package

```
9352 \bool_if:NTF \g_@@_messages_for_Overleaf_bool
9353 { \str_const:Nn \c_@@_available_keys_str { } }
9354 {
9355   \str_const:Nn \c_@@_available_keys_str
9356   { For~a~list~of~the~available~keys,~type-H~<return>. }
9357 }
9358 \seq_new:N \g_@@_types_of_matrix_seq
9359 \seq_gset_from_clist:Nn \g_@@_types_of_matrix_seq
9360 {
9361   NiceMatrix ,
9362   pNiceMatrix , bNiceMatrix , vNiceMatrix, BNiceMatrix, VNiceMatrix
9363 }
9364 \seq_gset_map_x:NNn \g_@@_types_of_matrix_seq \g_@@_types_of_matrix_seq
9365 { \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:NoTF` 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`.

```
9366 \cs_new_protected:Npn \@@_error_too_much_cols:
9367 {
9368   \seq_if_in:NoTF \g_@@_types_of_matrix_seq \g_@@_name_env_str
9369   {
9370     \int_compare:nNnTF \l_@@_last_col_int = { -2 }
9371     { \@@_fatal:n { too~much~cols~for~matrix } }
9372     {
9373       \int_compare:nNnTF \l_@@_last_col_int = { -1 }
9374       { \@@_fatal:n { too~much~cols~for~matrix } }
9375       {
9376         \bool_if:NF \l_@@_last_col_without_value_bool
9377         { \@@_fatal:n { too~much~cols~for~matrix~with~last~col } }
9378       }
9379     }
9380   }
9381   { \@@_fatal:nn { too~much~cols~for~array } }
9382 }
```

The following command must *not* be protected since it's used in an error message.

```

9383 \cs_new:Npn \@@_message_hdotsfor:
9384 {
9385   \tl_if_empty:oF \g_@@_HVDotsfor_lines_tl
9386   { ~Maybe~your~use~of~\token_to_str:N \Hdotsfor\ is~incorrect.}
9387 }
9388 \@@_msg_new:nn { hvlines,~rounded-corners~and~corners }
9389 {
9390   Incompatible~options.\\
9391   You~should~not~use~'hvlines',~'rounded-corners'~and~'corners'~at~this~time.\\
9392   The~output~will~not~be~reliable.
9393 }
9394 \@@_msg_new:nn { negative~weight }
9395 {
9396   Negative~weight.\\
9397   The~weight~of~the~'X'~columns~must~be~positive~and~you~have~used~
9398   the~value~'\int_use:N \l_@@_weight_int'.\\
9399   The~absolute~value~will~be~used.
9400 }
9401 \@@_msg_new:nn { last~col~not~used }
9402 {
9403   Column~not~used.\\
9404   The~key~'last~col'~is~in~force~but~you~have~not~used~that~last~column~
9405   in~your~\@@_full_name_env:.~However,~you~can~go~on.
9406 }
9407 \@@_msg_new:nn { too~much~cols~for~matrix~with~last~col }
9408 {
9409   Too~much~columns.\\
9410   In~the~row~\int_eval:n { \c@iRow },~
9411   you~try~to~use~more~columns~
9412   than~allowed~by~your~\@@_full_name_env:.~\@@_message_hdotsfor:\
9413   The~maximal~number~of~columns~is~\int_eval:n { \l_@@_last_col_int - 1 }~
9414   (plus~the~exterior~columns).~This~error~is~fatal.
9415 }
9416 \@@_msg_new:nn { too~much~cols~for~matrix }
9417 {
9418   Too~much~columns.\\
9419   In~the~row~\int_eval:n { \c@iRow },~
9420   you~try~to~use~more~columns~than~allowed~by~your~
9421   \@@_full_name_env:.~\@@_message_hdotsfor:\ Recall~that~the~maximal~
9422   number~of~columns~for~a~matrix~(excepted~the~potential~exterior~
9423   columns)~is~fixed~by~the~LaTeX~counter~'MaxMatrixCols'.~
9424   Its~current~value~is~\int_use:N \c@MaxMatrixCols\ (use~
9425   \token_to_str:N \setcounter\ to~change~that~value).~
9426   This~error~is~fatal.
9427 }
9428 \@@_msg_new:nn { too~much~cols~for~array }
9429 {
9430   Too~much~columns.\\
9431   In~the~row~\int_eval:n { \c@iRow },~
9432   ~you~try~to~use~more~columns~than~allowed~by~your~
9433   \@@_full_name_env:.~\@@_message_hdotsfor:\ The~maximal~number~of~columns~is~
9434   \int_use:N \g_@@_static_num_of_col_int\
9435   ~(plus~the~potential~exterior~ones).~
9436   This~error~is~fatal.
9437 }
9438 \@@_msg_new:nn { columns~not~used }
9439 {
9440   Columns~not~used.\\
9441   The~preamble~of~your~\@@_full_name_env:\ announces~\int_use:N

```

```

9442 \g_@@_static_num_of_col_int\ columns~but~you~use~only~\int_use:N \c@jCol.\
9443 The~columns~you~did~not~used~won't~be~created.\
9444 You~won't~have~similar~error~message~till~the~end~of~the~document.
9445 }
9446 \@@_msg_new:nn { empty~preamble }
9447 {
9448   Empty~preamble.\
9449   The~preamble~of~your~\@@_full_name_env:\ is~empty.\
9450   This~error~is~fatal.
9451 }
9452 \@@_msg_new:nn { in~first~col }
9453 {
9454   Erroneous~use.\
9455   You~can't~use~the~command~#1 in~the~first~column~(number~0)~of~the~array.\
9456   That~command~will~be~ignored.
9457 }
9458 \@@_msg_new:nn { in~last~col }
9459 {
9460   Erroneous~use.\
9461   You~can't~use~the~command~#1 in~the~last~column~(exterior)~of~the~array.\
9462   That~command~will~be~ignored.
9463 }
9464 \@@_msg_new:nn { in~first~row }
9465 {
9466   Erroneous~use.\
9467   You~can't~use~the~command~#1 in~the~first~row~(number~0)~of~the~array.\
9468   That~command~will~be~ignored.
9469 }
9470 \@@_msg_new:nn { in~last~row }
9471 {
9472   You~can't~use~the~command~#1 in~the~last~row~(exterior)~of~the~array.\
9473   That~command~will~be~ignored.
9474 }
9475 \@@_msg_new:nn { caption~outside~float }
9476 {
9477   Key~caption~forbidden.\
9478   You~can't~use~the~key~'caption'~because~you~are~not~in~a~floating~
9479   environment.~This~key~will~be~ignored.
9480 }
9481 \@@_msg_new:nn { short~caption~without~caption }
9482 {
9483   You~should~not~use~the~key~'short~caption'~without~'caption'.~
9484   However,~your~'short~caption'~will~be~used~as~'caption'.
9485 }
9486 \@@_msg_new:nn { double~closing~delimiter }
9487 {
9488   Double~delimiter.\
9489   You~can't~put~a~second~closing~delimiter~"#1"~just~after~a~first~closing~
9490   delimiter.~This~delimiter~will~be~ignored.
9491 }
9492 \@@_msg_new:nn { delimiter~after~opening }
9493 {
9494   Double~delimiter.\
9495   You~can't~put~a~second~delimiter~"#1"~just~after~a~first~opening~
9496   delimiter.~That~delimiter~will~be~ignored.
9497 }
9498 \@@_msg_new:nn { bad~option~for~line~style }
9499 {
9500   Bad~line~style.\

```

```

9501     Since~you~haven't~loaded~Tikz,~the~only~value~you~can~give~to~'line-style'~
9502     is~'standard'.~That~key~will~be~ignored.
9503 }
9504 \@@_msg_new:nn { Identical~notes~in~caption }
9505 {
9506     Identical~tabular~notes.\\
9507     You~can't~put~several~notes~with~the~same~content~in~
9508     \token_to_str:N \caption\ (but~you~can~in~the~main~tabular).\\
9509     If~you~go~on,~the~output~will~probably~be~erroneous.
9510 }
9511 \@@_msg_new:nn { tabularnote~below~the~tabular }
9512 {
9513     \token_to_str:N \tabularnote\ forbidden\\
9514     You~can't~use~\token_to_str:N \tabularnote\ in~the~caption~
9515     of~your~tabular~because~the~caption~will~be~composed~below~
9516     the~tabular.~If~you~want~the~caption~above~the~tabular~use~the~
9517     key~'caption~above'~in~\token_to_str:N \NiceMatrixOptions.\\
9518     Your~\token_to_str:N \tabularnote\ will~be~discarded~and~
9519     no~similar~error~will~raised~in~this~document.
9520 }
9521 \@@_msg_new:nn { Unknown~key~for~rules }
9522 {
9523     Unknown~key.\\
9524     There~is~only~two~keys~available~here:~width~and~color.\\
9525     Your~key~'\l_keys_key_str'~will~be~ignored.
9526 }
9527 \@@_msg_new:nn { Unknown~key~for~TikzEveryCell }
9528 {
9529     Unknown~key.\\
9530     There~is~only~two~keys~available~here:~
9531     'empty'~and~'not-empty'.\\
9532     Your~key~'\l_keys_key_str'~will~be~ignored.
9533 }
9534 \@@_msg_new:nn { Unknown~key~for~rotate }
9535 {
9536     Unknown~key.\\
9537     The~only~key~available~here~is~'c'.\\
9538     Your~key~'\l_keys_key_str'~will~be~ignored.
9539 }
9540 \@@_msg_new:nnn { Unknown~key~for~custom~line }
9541 {
9542     Unknown~key.\\
9543     The~key~'\l_keys_key_str'~is~unknown~in~a~'custom~line'.~
9544     It~you~go~on,~you~will~probably~have~other~errors. \\
9545     \c_@@_available_keys_str
9546 }
9547 {
9548     The~available~keys~are~(in~alphabetic~order):~
9549     ccommand,~
9550     color,~
9551     command,~
9552     dotted,~
9553     letter,~
9554     multiplicity,~
9555     sep-color,~
9556     tikz,~and~total-width.
9557 }
9558 \@@_msg_new:nnn { Unknown~key~for~xdots }
9559 {
9560     Unknown~key.\\
9561     The~key~'\l_keys_key_str'~is~unknown~for~a~command~for~drawing~dotted~rules.\\

```

```

9562     \c_@@_available_keys_str
9563   }
9564   {
9565     The-available-keys-are-(in-alphabetic-order):~
9566     'color',~
9567     'horizontal-labels',~
9568     'inter',~
9569     'line-style',~
9570     'radius',~
9571     'shorten',~
9572     'shorten-end'~and~'shorten-start'.
9573   }

9574 \@@_msg_new:nn { Unknown~key~for~rowcolors }
9575   {
9576     Unknown~key.\\
9577     As~for~now,~there-is~only~two~keys~available~here:~'cols'~and~'respect-blocks'~
9578     (and~you~try~to~use~'\l_keys_key_str')\\
9579     That~key~will~be~ignored.
9580   }

9581 \@@_msg_new:nn { label~without~caption }
9582   {
9583     You~can't~use~the~key~'label'~in~your~'{NiceTabular}'~because~
9584     you~have~not~used~the~key~'caption'.~The~key~'label'~will~be~ignored.
9585   }

9586 \@@_msg_new:nn { W~warning }
9587   {
9588     Line~\msg_line_number:~The~cell~is~too~wide~for~your~column~'W'~
9589     (row~\int_use:N \c@iRow).
9590   }

9591 \@@_msg_new:nn { Construct~too~large }
9592   {
9593     Construct~too~large.\\
9594     Your~command~\token_to_str:N #1
9595     can't~be~drawn~because~your~matrix~is~too~small.\\
9596     That~command~will~be~ignored.
9597   }

9598 \@@_msg_new:nn { underscore~after~nicematrix }
9599   {
9600     Problem~with~'underscore'.\\
9601     The~package~'underscore'~should~be~loaded~before~'nicematrix'.~
9602     You~can~go~on~but~you~won't~be~able~to~write~something~such~as:\\
9603     '\token_to_str:N \Cdots\token_to_str:N _{n~\token_to_str:N \text{~times}}'.
9604   }

9605 \@@_msg_new:nn { ampersand~in~light-syntax }
9606   {
9607     Ampersand~forbidden.\\
9608     You~can't~use~an~ampersand~(\token_to_str:N &)~to~separate~columns~because~
9609     the~key~'light-syntax'~is~in~force.~This~error~is~fatal.
9610   }

9611 \@@_msg_new:nn { double~backslash~in~light-syntax }
9612   {
9613     Double~backslash~forbidden.\\
9614     You~can't~use~\token_to_str:N
9615     \\~to~separate~rows~because~the~key~'light-syntax'~
9616     is~in~force.~You~must~use~the~character~'\l_@@_end_of_row_tl'~
9617     (set~by~the~key~'end-of-row').~This~error~is~fatal.
9618   }

9619 \@@_msg_new:nn { hlines~with~color }
9620   {
9621     Incompatible~keys.\\

```

```

9622     You~can't~use~the~keys~'hlines',~'vlines'~or~'hvlines'~for~a~
9623     '\token_to_str:N \Block'~when~the~key~'color'~or~'draw'~is~used.\\
9624     However,~you~can~put~several~commands~\token_to_str:N \Block.\\
9625     Your~key~will~be~discarded.
9626 }
9627 \@@_msg_new:nn { bad~value~for~baseline }
9628 {
9629     Bad~value~for~baseline.\\
9630     The~value~given~to~'baseline'~(\int_use:N \l_tmpa_int)~is~not~
9631     valid.~The~value~must~be~between~\int_use:N \l_@@_first_row_int\ and~
9632     \int_use:N \g_@@_row_total_int\ or~equal~to~'t',~'c'~or~'b'~or~of~
9633     the~form~'line-i'.\\
9634     A~value~of~1~will~be~used.
9635 }
9636 \@@_msg_new:nn { detection~of~empty~cells }
9637 {
9638     Problem~with~'not~empty'\\
9639     For~technical~reasons,~you~must~activate~
9640     'create~cell~nodes'~in~\token_to_str:N \CodeBefore\
9641     in~order~to~use~the~key~'\l_keys_key_str'.\\
9642     That~key~will~be~ignored.
9643 }
9644 \@@_msg_new:nn { siunitx~not~loaded }
9645 {
9646     siunitx~not~loaded\\
9647     You~can't~use~the~columns~'S'~because~'siunitx'~is~not~loaded.\\
9648     That~error~is~fatal.
9649 }
9650 \@@_msg_new:nn { ragged2e~not~loaded }
9651 {
9652     You~have~to~load~'ragged2e'~in~order~to~use~the~key~'\l_keys_key_str'~in~
9653     your~column~'\l_@@_vpos_col_str'~(or~'X').~The~key~'\str_lowercase:o
9654     \l_keys_key_str'~will~be~used~instead.
9655 }
9656 \@@_msg_new:nn { Invalid~name }
9657 {
9658     Invalid~name.\\
9659     You~can't~give~the~name~'\l_keys_value_tl'~to~a~\token_to_str:N
9660     \SubMatrix\ of~your~\@@_full_name_env:.\\
9661     A~name~must~be~accepted~by~the~regular~expression~[A-Za-z][A-Za-z0-9]*.\\
9662     This~key~will~be~ignored.
9663 }
9664 \@@_msg_new:nn { Wrong~line~in~SubMatrix }
9665 {
9666     Wrong~line.\\
9667     You~try~to~draw~a~#1~line~of~number~'#2'~in~a~
9668     \token_to_str:N \SubMatrix\ of~your~\@@_full_name_env:\ but~that~
9669     number~is~not~valid.~It~will~be~ignored.
9670 }
9671 \@@_msg_new:nn { Impossible~delimiter }
9672 {
9673     Impossible~delimiter.\\
9674     It's~impossible~to~draw~the~#1~delimiter~of~your~
9675     \token_to_str:N \SubMatrix\ because~all~the~cells~are~empty~
9676     in~that~column.
9677     \bool_if:NT \l_@@_submatrix_slim_bool
9678     { ~Maybe~you~should~try~without~the~key~'slim'. } \\
9679     This~\token_to_str:N \SubMatrix\ will~be~ignored.
9680 }
9681 \@@_msg_new:nnn { width~without~X~columns }

```

```

9682 {
9683     You-have-used-the-key~'width'~but-you-have-put-no~'X'~column.~
9684     That-key-will-be-ignored.
9685 }
9686 {
9687     This-message-is~the-message~'width-without-X-columns'~
9688     of~the-module~'nicematrix'.~
9689     The-experimented-users-can-disable-that-message-with~
9690     \token_to_str:N \msg_redirect_name:nnn.\\
9691 }
9692
9693 \@@_msg_new:nn { key-multiplicity-with-dotted }
9694 {
9695     Incompatible-keys. \\
9696     You-have-used-the-key~'multiplicity'~with-the-key~'dotted'~
9697     in~a~'custom-line'~.They-are-incompatible. \\
9698     The-key~'multiplicity'~will-be-discarded.
9699 }
9700 \@@_msg_new:nn { empty-environment }
9701 {
9702     Empty-environment.\\
9703     Your~\@@_full_name_env:\ is-empty.~This-error-is-fatal.
9704 }
9705 \@@_msg_new:nn { No-letter-and-no-command }
9706 {
9707     Erroneous-use.\\
9708     Your-use-of~'custom-line'~is-no-op~since-you-don't-have-used-the~
9709     key~'letter'~(for-a-letter-for-vertical-rules)~nor-the-keys~'command'~or~
9710     ~'ccommand'~(to-draw-horizontal-rules).\\
9711     However,~you-can-go-on.
9712 }
9713 \@@_msg_new:nn { Forbidden-letter }
9714 {
9715     Forbidden-letter.\\
9716     You-can't-use-the-letter~'#1'~for-a-customized-line.\\
9717     It-will-be-ignored.
9718 }
9719 \@@_msg_new:nn { Several-letters }
9720 {
9721     Wrong-name.\\
9722     You-must-use-only-one-letter-as-value-for-the-key~'letter'~(and-you~
9723     have-used~'\l_@@_letter_str').\\
9724     It-will-be-ignored.
9725 }
9726 \@@_msg_new:nn { Delimiter-with-small }
9727 {
9728     Delimiter~forbidden.\\
9729     You-can't-put-a-delimiter~in~the-preamble-of~your~\@@_full_name_env:\
9730     because-the-key~'small'~is-in-force.\\
9731     This-error-is-fatal.
9732 }
9733 \@@_msg_new:nn { unknown-cell-for-line-in-CodeAfter }
9734 {
9735     Unknown-cell.\\
9736     Your-command~\token_to_str:N\line\{#1\}\{#2\}~in~
9737     the~\token_to_str:N \CodeAfter\ of~your~\@@_full_name_env:\
9738     can't-be-executed-because-a-cell-doesn't-exist.\\
9739     This-command~\token_to_str:N \line\ will-be-ignored.
9740 }
9741 \@@_msg_new:nnn { Duplicate-name-for-SubMatrix }

```



```

9742 {
9743   Duplicate-name.\
9744   The-name-'\#1'-is-already-used-for-a-'\token_to_str:N \SubMatrix\
9745   in-this-'\@@_full_name_env:.\
9746   This-key-will-be-ignored.\
9747   \bool_if:NF \g_@@_messages_for_Overleaf_bool
9748   { For-a-list-of-the-names-already-used,~type-H<return>. }
9749 }
9750 {
9751   The-names-already-defined-in-this-'\@@_full_name_env:\ are:~
9752   \seq_use:Nnnn \g_@@_submatrix_names_seq { ~and~ } { ,~ } { ~and~ }.
9753 }
9754 \@@_msg_new:nn { r-or-l-with-preamble }
9755 {
9756   Erroneous-use.\
9757   You-can't-use-the-key-'\l_keys_key_str'~in-your-'\@@_full_name_env:~
9758   You-must-specify-the-alignment-of-your-columns-with-the-preamble-of-
9759   your-'\@@_full_name_env:.\
9760   This-key-will-be-ignored.
9761 }
9762 \@@_msg_new:nn { Hdotsfor~in~col-0 }
9763 {
9764   Erroneous-use.\
9765   You-can't-use-'\token_to_str:N \Hdotsfor\ in~an-exterior~column~of~
9766   the-array.~This-error-is-fatal.
9767 }
9768 \@@_msg_new:nn { bad-corner }
9769 {
9770   Bad-corner.\
9771   #1-is-an-incorrect-specification-for-a-corner~(in~the-key~
9772   'corners').~The-available-values-are:~NW,~SW,~NE~and~SE.\
9773   This-specification-of~corner~will-be-ignored.
9774 }
9775 \@@_msg_new:nn { bad-border }
9776 {
9777   Bad-border.\
9778   \l_keys_key_str\space-is-an-incorrect-specification-for-a-border~
9779   (in~the-key~'borders'~of~the-command-'\token_to_str:N \Block).~
9780   The-available-values-are:~left,~right,~top~and~bottom~(and~you~can~
9781   also~use~the-key~'tikz'
9782   \IfPackageLoadedTF { tikz }
9783   { }
9784   {~if~you~load~the~LaTeX~package~'tikz'}).\\
9785   This-specification-of~border~will-be-ignored.
9786 }
9787 \@@_msg_new:nn { TikzEveryCell~without~tikz }
9788 {
9789   TikZ-not-loaded.\
9790   You-can't-use-'\token_to_str:N \TikzEveryCell\
9791   because~you~have~not~loaded~tikz.~
9792   This-command-will-be-ignored.
9793 }
9794 \@@_msg_new:nn { tikz-key~without~tikz }
9795 {
9796   TikZ-not-loaded.\
9797   You-can't-use~the~key~'tikz'~for~the~command~'\token_to_str:N
9798   \Block'~because~you~have~not~loaded~tikz.~
9799   This-key-will-be-ignored.
9800 }
9801 \@@_msg_new:nn { last-col~non-empty~for~NiceArray }
9802 {

```

```

9803     Erroneous~use.\\
9804     In~the~\@@_full_name_env:,~you~must~use~the~key~
9805     'last-col'~without~value.\\
9806     However,~you~can~go~on~for~this~time~
9807     (the~value~'\l_keys_value_tl'~will~be~ignored).
9808 }
9809 \@@_msg_new:nn { last-col~non-empty~for~NiceMatrixOptions }
9810 {
9811     Erroneous~use.\\
9812     In~\token_to_str:N \NiceMatrixOptions,~you~must~use~the~key~
9813     'last-col'~without~value.\\
9814     However,~you~can~go~on~for~this~time~
9815     (the~value~'\l_keys_value_tl'~will~be~ignored).
9816 }
9817 \@@_msg_new:nn { Block~too~large~1 }
9818 {
9819     Block~too~large.\\
9820     You~try~to~draw~a~block~in~the~cell~#1-#2~of~your~matrix~but~the~matrix~is~
9821     too~small~for~that~block. \\
9822     This~block~and~maybe~others~will~be~ignored.
9823 }
9824 \@@_msg_new:nn { Block~too~large~2 }
9825 {
9826     Block~too~large.\\
9827     The~preamble~of~your~\@@_full_name_env:\ announces~\int_use:N
9828     \g_@@_static_num_of_col_int\
9829     columns~but~you~use~only~\int_use:N \c@jCol\ and~that's~why~a~block~
9830     specified~in~the~cell~#1-#2~can't~be~drawn.~You~should~add~some~ampersands~
9831     (&)~at~the~end~of~the~first~row~of~your~\@@_full_name_env:.\
9832     This~block~and~maybe~others~will~be~ignored.
9833 }
9834 \@@_msg_new:nn { unknown~column~type }
9835 {
9836     Bad~column~type.\\
9837     The~column~type~'#1'~in~your~\@@_full_name_env:\
9838     is~unknown. \\
9839     This~error~is~fatal.
9840 }
9841 \@@_msg_new:nn { unknown~column~type~S }
9842 {
9843     Bad~column~type.\\
9844     The~column~type~'S'~in~your~\@@_full_name_env:\ is~unknown. \\
9845     If~you~want~to~use~the~column~type~'S'~of~siunitx,~you~should~
9846     load~that~package. \\
9847     This~error~is~fatal.
9848 }
9849 \@@_msg_new:nn { tabularnote~forbidden }
9850 {
9851     Forbidden~command.\\
9852     You~can't~use~the~command~\token_to_str:N\tabularnote\
9853     ~here.~This~command~is~available~only~in~
9854     \{NiceTabular\},~\{NiceTabular*\}~and~\{NiceTabularX\}~or~in~
9855     the~argument~of~a~command~\token_to_str:N \caption\ included~
9856     in~an~environment~{table}. \\
9857     This~command~will~be~ignored.
9858 }
9859 \@@_msg_new:nn { borders~forbidden }
9860 {
9861     Forbidden~key.\\
9862     You~can't~use~the~key~'borders'~of~the~command~\token_to_str:N \Block\
9863     because~the~option~'rounded-corners'~

```

```

9864     is~in~force~with~a~non~zero~value.\\
9865     This~key~will~be~ignored.
9866 }

9867 \@@_msg_new:nn { bottomrule~without~booktabs }
9868 {
9869     booktabs~not~loaded.\\
9870     You~can't~use~the~key~'tabular/bottomrule'~because~you~haven't~
9871     loaded~'booktabs'.\\
9872     This~key~will~be~ignored.
9873 }

9874 \@@_msg_new:nn { enumitem~not~loaded }
9875 {
9876     enumitem~not~loaded.\\
9877     You~can't~use~the~command~\token_to_str:N\tabularnote\
9878     ~because~you~haven't~loaded~'enumitem'.\\
9879     All~the~commands~\token_to_str:N\tabularnote\ will~be~
9880     ignored~in~the~document.
9881 }

9882 \@@_msg_new:nn { tikz~without~tikz }
9883 {
9884     Tikz~not~loaded.\\
9885     You~can't~use~the~key~'tikz'~here~because~Tikz~is~not~
9886     loaded.~If~you~go~on,~that~key~will~be~ignored.
9887 }

9888 \@@_msg_new:nn { tikz~in~custom~line~without~tikz }
9889 {
9890     Tikz~not~loaded.\\
9891     You~have~used~the~key~'tikz'~in~the~definition~of~a~
9892     customized~line~(with~'custom~line')~but~tikz~is~not~loaded.~
9893     You~can~go~on~but~you~will~have~another~error~if~you~actually~
9894     use~that~custom~line.
9895 }

9896 \@@_msg_new:nn { tikz~in~borders~without~tikz }
9897 {
9898     Tikz~not~loaded.\\
9899     You~have~used~the~key~'tikz'~in~a~key~'borders'~(of~a~
9900     command~\token_to_str:N\Block')~but~tikz~is~not~loaded.~
9901     That~key~will~be~ignored.
9902 }

9903 \@@_msg_new:nn { without~color~inside }
9904 {
9905     If~order~to~use~\token_to_str:N \cellcolor,~\token_to_str:N \rowcolor,~
9906     \token_to_str:N \rowcolors\ or~\token_to_str:N \rowlistcolors\
9907     outside~\token_to_str:N \CodeBefore,~you~
9908     should~have~used~the~key~'color~inside'~in~your~\@@_full_name_env:.\\
9909     You~can~go~on~but~you~may~need~more~compilations.
9910 }

9911 \@@_msg_new:nn { color~in~custom~line~with~tikz }
9912 {
9913     Erroneous~use.\\
9914     In~a~'custom~line',~you~have~used~both~'tikz'~and~'color',~
9915     which~is~forbidden~(you~should~use~'color'~inside~the~key~'tikz').~
9916     The~key~'color'~will~be~discarded.
9917 }

9918 \@@_msg_new:nn { Wrong~last~row }
9919 {
9920     Wrong~number.\\
9921     You~have~used~'last~row=\int_use:N \l_@@_last_row_int'~but~your~
9922     \@@_full_name_env:\ seems~to~have~\int_use:N \c@iRow \ rows.~
9923     If~you~go~on,~the~value~of~\int_use:N \c@iRow \ will~be~used~for~

```

```

9924     last-row.~You-can-avoid-this-problem-by-using~'last-row'~
9925     without~value~(more~compilations-might-be-necessary).
9926 }
9927 \@@_msg_new:nn { Yet~in~env }
9928 {
9929     Nested~environments.\\
9930     Environments-of~nicematrix~can't-be-nested.\\
9931     This~error~is~fatal.
9932 }
9933 \@@_msg_new:nn { Outside~math~mode }
9934 {
9935     Outside~math~mode.\\
9936     The~\@@_full_name_env:\ can-be-used-only~in~math~mode~
9937     (and~not~in~\token_to_str:N \vcenter).\\
9938     This~error~is~fatal.
9939 }
9940 \@@_msg_new:nn { One~letter~allowed }
9941 {
9942     Bad~name.\\
9943     The~value~of~key~'\l_keys_key_str'~must~be~of~length~1.\\
9944     It~will~be~ignored.
9945 }
9946 \@@_msg_new:nn { TabularNote~in~CodeAfter }
9947 {
9948     Environment~{TabularNote}~forbidden.\\
9949     You~must~use~{TabularNote}~at~the~end~of~your~{NiceTabular}~
9950     but~*before*~the~\token_to_str:N \CodeAfter.\\
9951     This~environment~{TabularNote}~will~be~ignored.
9952 }
9953 \@@_msg_new:nn { varwidth~not~loaded }
9954 {
9955     varwidth~not~loaded.\\
9956     You~can't~use~the~column~type~'V'~because~'varwidth'~is~not~
9957     loaded.\\
9958     Your~column~will~behave~like~'p'.
9959 }
9960 \@@_msg_new:nnn { Unknow~key~for~RulesBis }
9961 {
9962     Unknow~key.\\
9963     Your~key~'\l_keys_key_str'~is~unknown~for~a~rule.\\
9964     \c_@@_available_keys_str
9965 }
9966 {
9967     The~available~keys~are~(in~alphabetic~order):~
9968     color,~
9969     dotted,~
9970     multiplicity,~
9971     sep-color,~
9972     tikz,~and~total~width.
9973 }
9974
9975 \@@_msg_new:nnn { Unknown~key~for~Block }
9976 {
9977     Unknown~key.\\
9978     The~key~'\l_keys_key_str'~is~unknown~for~the~command~\token_to_str:N
9979     \Block.\\ It~will~be~ignored. \\
9980     \c_@@_available_keys_str
9981 }
9982 {
9983     The~available~keys~are~(in~alphabetic~order):~&~in~blocks,~ampersand~in~blocks,~
9984     b,~B,~borders,~c,~draw,~fill,~hlines,~hvlines,~l,~line~width,~name,~

```

```

9985     opacity,~rounded-corners,~r,~respect-arraystretch,~t,~T,~tikz,~transparent~
9986     and~vlines.
9987 }

9988 \@@_msg_new:nnn { Unknown~key~for~Brace }
9989 {
9990     Unknown~key.\\
9991     The~key~'\l_keys_key_str'~is~unknown~for~the~commands~\token_to_str:N
9992     \UnderBrace\ and~\token_to_str:N \OverBrace.\\
9993     It~will~be~ignored. \\
9994     \c_@@_available_keys_str
9995 }
9996 {
9997     The~available~keys~are~(in~alphabetic~order):~color,~left~shorten,~
9998     right~shorten,~shorten~(which~fixes~both~left~shorten~and~
9999     right~shorten)~and~yshift.
10000 }

10001 \@@_msg_new:nnn { Unknown~key~for~CodeAfter }
10002 {
10003     Unknown~key.\\
10004     The~key~'\l_keys_key_str'~is~unknown.\\
10005     It~will~be~ignored. \\
10006     \c_@@_available_keys_str
10007 }
10008 {
10009     The~available~keys~are~(in~alphabetic~order):~
10010     delimiters/color,~
10011     rules~(with~the~subkeys~'color'~and~'width'),~
10012     sub~matrix~(several~subkeys)~
10013     and~xdots~(several~subkeys).~
10014     The~latter~is~for~the~command~\token_to_str:N \line.
10015 }

10016 \@@_msg_new:nnn { Unknown~key~for~CodeBefore }
10017 {
10018     Unknown~key.\\
10019     The~key~'\l_keys_key_str'~is~unknown.\\
10020     It~will~be~ignored. \\
10021     \c_@@_available_keys_str
10022 }
10023 {
10024     The~available~keys~are~(in~alphabetic~order):~
10025     create~cell~nodes,~
10026     delimiters/color~and~
10027     sub~matrix~(several~subkeys).
10028 }

10029 \@@_msg_new:nnn { Unknown~key~for~SubMatrix }
10030 {
10031     Unknown~key.\\
10032     The~key~'\l_keys_key_str'~is~unknown.\\
10033     That~key~will~be~ignored. \\
10034     \c_@@_available_keys_str
10035 }
10036 {
10037     The~available~keys~are~(in~alphabetic~order):~
10038     'delimiters/color',~
10039     'extra~height',~
10040     'hlines',~
10041     'hvlines',~
10042     'left~xshift',~
10043     'name',~
10044     'right~xshift',~
10045     'rules'~(with~the~subkeys~'color'~and~'width'),~
10046     'slim',~

```

```

10047     'vlines'~and~'xshift'~(which~sets~both~'left-xshift'~
10048     and~'right-xshift').\\
10049 }
10050 \@@_msg_new:nnn { Unknown~key~for~notes }
10051 {
10052     Unknown~key.\\
10053     The~key~'\l_keys_key_str'~is~unknown.\\
10054     That~key~will~be~ignored. \\
10055     \c_@@_available_keys_str
10056 }
10057 {
10058     The~available~keys~are~(in~alphabetic~order):~
10059     bottomrule,~
10060     code~after,~
10061     code~before,~
10062     detect~duplicates,~
10063     enumitem~keys,~
10064     enumitem~keys~para,~
10065     para,~
10066     label~in~list,~
10067     label~in~tabular~and~
10068     style.
10069 }
10070 \@@_msg_new:nnn { Unknown~key~for~RowStyle }
10071 {
10072     Unknown~key.\\
10073     The~key~'\l_keys_key_str'~is~unknown~for~the~command~
10074     \token_to_str:N \RowStyle. \\
10075     That~key~will~be~ignored. \\
10076     \c_@@_available_keys_str
10077 }
10078 {
10079     The~available~keys~are~(in~alphabetic~order):~
10080     'bold',~
10081     'cell-space-top-limit',~
10082     'cell-space-bottom-limit',~
10083     'cell-space-limits',~
10084     'color',~
10085     'nb-rows'~and~
10086     'rowcolor'.
10087 }
10088 \@@_msg_new:nnn { Unknown~key~for~NiceMatrixOptions }
10089 {
10090     Unknown~key.\\
10091     The~key~'\l_keys_key_str'~is~unknown~for~the~command~
10092     \token_to_str:N \NiceMatrixOptions. \\
10093     That~key~will~be~ignored. \\
10094     \c_@@_available_keys_str
10095 }
10096 {
10097     The~available~keys~are~(in~alphabetic~order):~
10098     &~in~blocks,~
10099     allow~duplicate~names,~
10100     ampersand~in~blocks,~
10101     caption~above,~
10102     cell~space~bottom~limit,~
10103     cell~space~limits,~
10104     cell~space~top~limit,~
10105     code~for~first~col,~
10106     code~for~first~row,~
10107     code~for~last~col,~
10108     code~for~last~row,~
10109     corners,~

```

```

10110 custom-key,~
10111 create-extra-nodes,~
10112 create-medium-nodes,~
10113 create-large-nodes,~
10114 custom-line,~
10115 delimiters~(several~subkeys),~
10116 end-of-row,~
10117 first-col,~
10118 first-row,~
10119 hlines,~
10120 hvlines,~
10121 hvlines-except-borders,~
10122 last-col,~
10123 last-row,~
10124 left-margin,~
10125 light-syntax,~
10126 light-syntax-expanded,~
10127 matrix/columns-type,~
10128 no-cell-nodes,~
10129 notes~(several~subkeys),~
10130 nullify-dots,~
10131 pgf-node-code,~
10132 renew-dots,~
10133 renew-matrix,~
10134 respect-arraystretch,~
10135 rounded-corners,~
10136 right-margin,~
10137 rules~(with~the~subkeys~'color'~and~'width'),~
10138 small,~
10139 sub-matrix~(several~subkeys),~
10140 vlimes,~
10141 xdots~(several~subkeys).
10142 }

```

For ‘{NiceArray}’, the set of keys is the same as for {NiceMatrix} excepted that there is no `l` and `r`.

```

10143 \@@_msg_new:nnn { Unknown~key~for~NiceArray }
10144 {
10145   Unknown~key.\\
10146   The~key~'\l_keys_key_str'~is~unknown~for~the~environment~
10147   \{NiceArray\}. \\
10148   That~key~will~be~ignored. \\
10149   \c_@@_available_keys_str
10150 }
10151 {
10152   The~available~keys~are~(in~alphabetic~order):~
10153   &-in-blocks,~
10154   ampersand-in-blocks,~
10155   b,~
10156   baseline,~
10157   c,~
10158   cell-space-bottom-limit,~
10159   cell-space-limits,~
10160   cell-space-top-limit,~
10161   code-after,~
10162   code-for-first-col,~
10163   code-for-first-row,~
10164   code-for-last-col,~
10165   code-for-last-row,~
10166   color-inside,~
10167   columns-width,~
10168   corners,~
10169   create-extra-nodes,~
10170   create-medium-nodes,~

```

```

10171     create-large-nodes,~
10172     extra-left-margin,~
10173     extra-right-margin,~
10174     first-col,~
10175     first-row,~
10176     hlines,~
10177     hvlines,~
10178     hvlines-except-borders,~
10179     last-col,~
10180     last-row,~
10181     left-margin,~
10182     light-syntax,~
10183     light-syntax-expanded,~
10184     name,~
10185     no-cell-nodes,~
10186     nullify-dots,~
10187     pgf-node-code,~
10188     renew-dots,~
10189     respect-arraystretch,~
10190     right-margin,~
10191     rounded-corners,~
10192     rules~(with~the~subkeys~'color'~and~'width'),~
10193     small,~
10194     t,~
10195     vlines,~
10196     xdots/color,~
10197     xdots/shorten-start,~
10198     xdots/shorten-end,~
10199     xdots/shorten-and~
10200     xdots/line-style.
10201 }

```

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`).

```

10202 \@@_msg_new:nnn { Unknown~key~for~NiceMatrix }
10203 {
10204   Unknown~key.\\
10205   The~key~'\l_keys_key_str'~is~unknown~for~the~
10206   \@@_full_name_env:. \\
10207   That~key~will~be~ignored. \\
10208   \c_@@_available_keys_str
10209 }
10210 {
10211   The~available~keys~are~(in~alphabetic~order):~
10212   &-in-blocks,~
10213   ampersand-in-blocks,~
10214   b,~
10215   baseline,~
10216   c,~
10217   cell-space-bottom-limit,~
10218   cell-space-limits,~
10219   cell-space-top-limit,~
10220   code-after,~
10221   code-for-first-col,~
10222   code-for-first-row,~
10223   code-for-last-col,~
10224   code-for-last-row,~
10225   color-inside,~
10226   columns-type,~
10227   columns-width,~
10228   corners,~
10229   create-extra-nodes,~
10230   create-medium-nodes,~
10231   create-large-nodes,~

```



```

10232     extra-left-margin,~
10233     extra-right-margin,~
10234     first-col,~
10235     first-row,~
10236     hlines,~
10237     hvlines,~
10238     hvlines-except-borders,~
10239     l,~
10240     last-col,~
10241     last-row,~
10242     left-margin,~
10243     light-syntax,~
10244     light-syntax-expanded,~
10245     name,~
10246     no-cell-nodes,~
10247     nullify-dots,~
10248     pgf-node-code,~
10249     r,~
10250     renew-dots,~
10251     respect-arraystretch,~
10252     right-margin,~
10253     rounded-corners,~
10254     rules~(with~the~subkeys~'color'~and~'width'),~
10255     small,~
10256     t,~
10257     vlines,~
10258     xdots/color,~
10259     xdots/shorten-start,~
10260     xdots/shorten-end,~
10261     xdots/shorten-and~
10262     xdots/line-style.
10263 }
10264 \@@_msg_new:nnn { Unknown~key~for~NiceTabular }
10265 {
10266   Unknown~key.\\
10267   The~key~'\l_keys_key_str'~is~unknown~for~the~environment~
10268   \{NiceTabular\}. \\
10269   That~key~will~be~ignored. \\
10270   \c_@@_available_keys_str
10271 }
10272 {
10273   The~available~keys~are~(in~alphabetic~order):~
10274   &-in-blocks,~
10275   ampersand-in-blocks,~
10276   b,~
10277   baseline,~
10278   c,~
10279   caption,~
10280   cell-space-bottom-limit,~
10281   cell-space-limits,~
10282   cell-space-top-limit,~
10283   code-after,~
10284   code-for-first-col,~
10285   code-for-first-row,~
10286   code-for-last-col,~
10287   code-for-last-row,~
10288   color-inside,~
10289   columns-width,~
10290   corners,~
10291   custom-line,~
10292   create-extra-nodes,~
10293   create-medium-nodes,~
10294   create-large-nodes,~

```

```

10295     extra-left-margin,~
10296     extra-right-margin,~
10297     first-col,~
10298     first-row,~
10299     hlines,~
10300     hvlines,~
10301     hvlines-except-borders,~
10302     label,~
10303     last-col,~
10304     last-row,~
10305     left-margin,~
10306     light-syntax,~
10307     light-syntax-expanded,~
10308     name,~
10309     no-cell-nodes,~
10310     notes~(several~subkeys),~
10311     nullify-dots,~
10312     pgf-node-code,~
10313     renew-dots,~
10314     respect-arraystretch,~
10315     right-margin,~
10316     rounded-corners,~
10317     rules~(with~the~subkeys~'color'~and~'width'),~
10318     short-caption,~
10319     t,~
10320     tabularnote,~
10321     vlines,~
10322     xdots/color,~
10323     xdots/shorten-start,~
10324     xdots/shorten-end,~
10325     xdots/shorten-and~
10326     xdots/line-style.
10327 }

10328 \@@_msg_new:nnn { Duplicate-name }
10329 {
10330   Duplicate-name.\
10331   The-name-'~\l_keys_value_tl'~is~already~used~and~you~shouldn't~use~
10332   the~same~environment~name~twice.~You~can~go~on,~but,~
10333   maybe,~you~will~have~incorrect~results~especially~
10334   if~you~use~'columns-width=auto'.~If~you~don't~want~to~see~this~
10335   message~again,~use~the~key~'allow-duplicate-names'~in~
10336   '\token_to_str:N \NiceMatrixOptions'~.\
10337   \bool_if:NF \g_@@_messages_for_Overleaf_bool
10338     { For~a~list~of~the~names~already~used,~type~H~<return>. }
10339 }
10340 {
10341   The~names~already~defined~in~this~document~are:~
10342   \seq_use:Nnnn \g_@@_names_seq { ~and~ } { ,~ } { ~and~ }.
10343 }

10344 \@@_msg_new:nn { Option-auto-for-columns-width }
10345 {
10346   Erroneous-use.\
10347   You~can't~give~the~value~'auto'~to~the~key~'columns-width'~here.~
10348   That~key~will~be~ignored.
10349 }

10350 \@@_msg_new:nn { NiceTabularX~without~X }
10351 {
10352   NiceTabularX~without~X.\
10353   You~should~not~use~{NiceTabularX}~without~X~columns.\
10354   However,~you~can~go~on.
10355 }

10356 \@@_msg_new:nn { Preamble~forgotten }

```

```
10357 {
10358     Preamble~forgotten.\\
10359     You~have~probably~forgotten~the~preamble~of~your~
10360     \@@_full_name_env:. \\
10361     This~error~is~fatal.
10362 }
```

Contents

1	Declaration of the package and packages loaded	1
2	Security test	3
3	Collecting options	4
4	Technical definitions	4
5	Parameters	9
6	The command <code>\tabularnote</code>	19
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	91
15	<code>{NiceTabular}</code> , <code>{NiceTabularX}</code> and <code>{NiceTabular*}</code>	92
16	After the construction of the array	94
17	We draw the dotted lines	100
18	The actual instructions for drawing the dotted lines with Tikz	114
19	User commands available in the new environments	119
20	The command <code>\line</code> accessible in code-after	125
21	The command <code>\RowStyle</code>	127
22	Colors of cells, rows and columns	129
23	The vertical and horizontal rules	142
24	The empty corners	156
25	The environment <code>{NiceMatrixBlock}</code>	159
26	The extra nodes	160
27	The blocks	164
28	How to draw the dotted lines transparently	187
29	Automatic arrays	187
30	The redefinition of the command <code>\dotfill</code>	189

31	The command <code>\diagbox</code>	189
32	The keyword <code>\CodeAfter</code>	191
33	The delimiters in the preamble	191
34	The command <code>\SubMatrix</code>	193
35	Les commandes <code>\UnderBrace</code> et <code>\OverBrace</code>	201
36	The command <code>TikzEveryCell</code>	204
37	The command <code>\ShowCellNames</code>	205
38	We process the options at package loading	208
39	About the package underscore	210
40	Error messages of the package	210