

The code of the package `nicematrix`*

F. Pantigny
fpantigny@wanadoo.fr

October 13, 2024

Abstract

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

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

1 Declaration of the package and packages loaded

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

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

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

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

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

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

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

```
14 \RequirePackage { amsmath }
```

```
15 \RequirePackage { array }
```

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

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

```

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

```

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

```

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

```

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

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

2 Security test

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

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

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

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

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

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

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

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

```

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

```

3 Collecting options

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

Exemple :

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

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

```

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

```

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

```

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

```

4 Technical definitions

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

```

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

```

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

```

153 \tl_new:N \l_@@_argspec_tl

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

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

```

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

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

```

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

```

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

```

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

```

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

Idem for `\CT@drs@`.

```

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

```

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

```

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

```

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

¹See question 99041 on TeX StackExchange.

```

263     \skip_horizontal:N \c_zero_dim
264 }

```

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

```

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

```

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

```

269 \cs_set:Npn \@@_cline

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

```

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

```



```

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

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

```

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

```

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

```

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

```

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

```

```

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

```

5 Parameters

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

```

333 \int_new:N \g_@@_env_int

```

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

```

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

```

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

```

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

```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```
348 \dim_new:N \l_@@_col_width_dim
349 \dim_set:Nn \l_@@_col_width_dim { -1 cm }
```

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

```
350 \int_new:N \g_@@_row_total_int
351 \int_new:N \g_@@_col_total_int
```

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

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

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

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

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

```
354 \tl_new:N \l_@@_hpos_cell_tl
355 \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_c_tl
```

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

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

Idem for the mono-row blocks.

```
357 \dim_new:N \g_@@_blocks_ht_dim
358 \dim_new:N \g_@@_blocks_dp_dim
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

For `\multicolumn`.

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

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

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

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

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

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

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

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

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

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

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

```
386 {
```

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

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

```
389   \fi:
```

```
390 }
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```
412 \int_new:N \l_@@_old_iRow_int
413 \int_new:N \l_@@_old_jCol_int
```

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

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

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

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

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

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

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

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

```
417 \bool_new:N \l_@@_X_columns_aux_bool
418 \dim_new:N \l_@@_X_columns_dim
```

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

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

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

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

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

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

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

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

```
422 \tl_new:N \l_@@_code_before_tl
423 \bool_new:N \l_@@_code_before_bool
```

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

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

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

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

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

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

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

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

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

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

```
438 \dim_new:N \g_@@_width_last_col_dim
439 \dim_new:N \g_@@_width_first_col_dim
```

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

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

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

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

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

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

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

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

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

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

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

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

If the user has used the key `corners`, all the cells which are in an (empty) corner will be stored in the following sequence.

```
444 \seq_new:N \l_@@_corners_cells_seq
```

Maybe we should use a `clist` instead of a `seq` here because we will frequently have to search elements in that sequence (and, in the `aux` file, it's writtent as a comma-separated list).

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

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

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

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

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

```
447 \seq_new:N \g_@@_multicolumn_cells_seq
```

```
448 \seq_new:N \g_@@_multicolumn_sizes_seq
```

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

```
449 \int_new:N \l_@@_row_min_int
```

```
450 \int_new:N \l_@@_row_max_int
```

```
451 \int_new:N \l_@@_col_min_int
```

```
452 \int_new:N \l_@@_col_max_int
```

The following counters will be used when drawing the rules.

```
453 \int_new:N \l_@@_start_int
```

```
454 \int_set_eq:NN \l_@@_start_int \c_one_int
```

```
455 \int_new:N \l_@@_end_int
```

```
456 \int_new:N \l_@@_local_start_int
```

```
457 \int_new:N \l_@@_local_end_int
```

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

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

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

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

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

```
460 \tl_new:N \l_@@_fill_tl
```

```
461 \tl_new:N \l_@@_opacity_tl
```

```
462 \tl_new:N \l_@@_draw_tl
```

```
463 \seq_new:N \l_@@_tikz_seq
```

```
464 \clist_new:N \l_@@_borders_clist
```

```
465 \dim_new:N \l_@@_rounded_corners_dim
```

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

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

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


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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Variables for the exterior rows and columns

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

- **First row**

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

```
490 \int_new:N \l_@@_first_row_int
491 \int_set:Nn \l_@@_first_row_int 1
```

- **First column**

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

```
492 \int_new:N \l_@@_first_col_int
493 \int_set_eq:NN \l_@@_first_col_int \c_one_int
```

- **Last row**

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

```
494 \int_new:N \l_@@_last_row_int
495 \int_set:Nn \l_@@_last_row_int { -2 }
```

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

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

Idem for `\l_@@_last_col_without_value_bool`

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

- **Last column**

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

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

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

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

```

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

```

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

```

500 \bool_new:N \g_@@_last_col_found_bool

```

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

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

```

501 \bool_new:N \l_@@_in_last_col_bool

```

Some utilities

```

502 \cs_set_protected:Npn \@@_cut_on_hyphen:w #1-#2\q_stop
503 {

```

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

```

504 \cs_set_nopar:Npn \l_tmpa_tl { #1 }
505 \cs_set_nopar:Npn \l_tmpb_tl { #2 }
506 }

```

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

```

507 \cs_new_protected:Npn \@@_expand_clist:N #1
508 {
509   \clist_if_in:Nof #1 \c_@@_all_tl
510   {
511     \clist_clear:N \l_tmpa_clist
512     \clist_map_inline:Nn #1
513     {

```

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

```

514       \tl_if_in:nnTF { ##1 } { - }
515       { \@@_cut_on_hyphen:w ##1 \q_stop }
516       {

```

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

```

517       \cs_set_nopar:Npn \l_tmpa_tl { ##1 }
518       \cs_set_nopar:Npn \l_tmpb_tl { ##1 }
519     }
520     \int_step_inline:nnn \l_tmpa_tl \l_tmpb_tl
521     { \clist_put_right:Nn \l_tmpa_clist { #####1 } }
522   }
523   \tl_set_eq:NN #1 \l_tmpa_clist
524 }
525 }

```

The following internal parameters are for:

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

```

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

```

6 The command `\tablarnote`

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

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

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

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

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

```
533 \int_new:N \g_@@_tablarnote_int
```

```
534 \cs_set:Npn \theHtablarnote { \int_use:N \g_@@_tablarnote_int }
```

```
535 \seq_new:N \g_@@_notes_seq
```

```
536 \seq_new:N \g_@@_notes_in_caption_seq
```

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

```
537 \tl_new:N \g_@@_tablarnote_tl
```

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

```
538 \seq_new:N \l_@@_notes_labels_seq
```

```
539 \newcounter{nicematrix_draft}
```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

```

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

```

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

```

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

```

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

```

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

```

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

$$\{label\}\{text\ of\ the\ tabularnote\}.$$

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

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

```

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

```

```

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

```

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

```

640     \hbox_set:Nn \l_tmpa_box
641     {

```

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

```

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

```

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

```

648     \int_gdecr:N \c@tabularnote
649     \int_set_eq:NN \l_tmpa_int \c@tabularnote

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

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

```

7 Command for creation of rectangle nodes

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

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


```

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

```

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

```

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

```

8 The options

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

```

746 \tl_new:N \l_@@_caption_tl
747 \tl_new:N \l_@@_short_caption_tl
748 \tl_new:N \l_@@_label_tl

```

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

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

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

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

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

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

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

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

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

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

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

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

```
755 \dim_new:N \l_@@_xdots_inter_dim
```

```
756 \hook_gput_code:nnn { begindocument } { . }
```

```
757 { \dim_set:Nn \l_@@_xdots_inter_dim { 0.45 em } }
```

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

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

```
758 \dim_new:N \l_@@_xdots_shorten_start_dim
```

```
759 \dim_new:N \l_@@_xdots_shorten_end_dim
```

```
760 \hook_gput_code:nnn { begindocument } { . }
```

```
761 {
```

```
762   \dim_set:Nn \l_@@_xdots_shorten_start_dim { 0.3 em }
```

```
763   \dim_set:Nn \l_@@_xdots_shorten_end_dim { 0.3 em }
```

```
764 }
```

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

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

```
765 \dim_new:N \l_@@_xdots_radius_dim
```

```
766 \hook_gput_code:nnn { begindocument } { . }
```

```
767 { \dim_set:Nn \l_@@_xdots_radius_dim { 0.53 pt } }
```

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

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

```
768 \tl_new:N \l_@@_xdots_line_style_tl
```

```
769 \tl_const:Nn \c_@@_standard_tl { standard }
```

```
770 \tl_set_eq:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl
```

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

```
771 \bool_new:N \l_@@_light_syntax_bool
772 \bool_new:N \l_@@_light_syntax_expanded_bool
```

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

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

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

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

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

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

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

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

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

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

```
780 \dim_new:N \l_@@_notes_above_space_dim
781 \hook_gput_code:nnn { begindocument } { . }
782 { \dim_set:Nn \l_@@_notes_above_space_dim { 1 mm } }
```

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

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

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

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

```
784 \cs_new_protected:Npn \@@_reset_arraystretch:
785 { \cs_set_nopar:Npn \arraystretch { 1 } }
```

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

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

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

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

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

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

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

```
789 \bool_new:N \l_@@_medium_nodes_bool
790 \bool_new:N \l_@@_large_nodes_bool
```

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

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

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

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

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

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

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

```
796 \tl_new:N \l_@@_end_of_row_tl
797 \tl_set:Nn \l_@@_end_of_row_tl { ; }
```

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

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

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

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

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

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

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

```

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

```

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

```

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

```

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

```

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

```

```

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

```

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

```

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

```

```

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

```

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

```

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

```

```

930 parallelize-diags .bool_set:N = \l_@@_parallelize_diags_bool ,

```

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

```

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

```

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

```

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

```

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

```

968 c .code:n = \tl_set:Nn \l_@@_baseline_tl c ,
969 t .code:n = \tl_set:Nn \l_@@_baseline_tl t ,
970 b .code:n = \tl_set:Nn \l_@@_baseline_tl b ,
971 baseline .tl_set:N = \l_@@_baseline_tl ,
972 baseline .value_required:n = true ,
973 columns-width .code:n =

```

We use `\str_if_eq:nnTF` which is slightly faster than `\tl_if_eq:nnTF`.

```

974 \str_if_eq:nnTF { #1 } { auto }
975 { \bool_set_true:N \l_@@_auto_columns_width_bool }
976 { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,
977 columns-width .value_required:n = true ,
978 name .code:n =

```

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

```

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

```


1040 }

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

```

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

```

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

```

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

```

```

1096      \int_zero:N \l_@@_last_col_int ,
1097      small .bool_set:N = \l_@@_small_bool ,
1098      small .value_forbidden:n = true ,

```

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

```

1099      renew-matrix .code:n = \@@_renew_matrix: ,
1100      renew-matrix .value_forbidden:n = true ,

```

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

```

1101      exterior-arraycolsep .bool_set:N = \l_@@_exterior_arraycolsep_bool ,

```

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

```

1102      columns-width .code:n =

```

We use `\str_if_eq:nnTF` which is slightly faster than `\tl_if_eq:nnTF`.

```

1103      \str_if_eq:nnTF { #1 } { auto }
1104      { \@@_error:n { Option~auto~for~columns-width } }
1105      { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,

```

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

```

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

```

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

```

1119 \NewDocumentCommand \NiceMatrixOptions { m }
1120 { \keys_set:nn { nicematrix / NiceMatrixOptions } { #1 } }

```

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

```

1121 \keys_define:nn { nicematrix / NiceMatrix }
1122 {
1123   last-col .code:n = \tl_if_empty:nTF { #1 }
1124   {
1125     \bool_set_true:N \l_@@_last_col_without_value_bool
1126     \int_set:Nn \l_@@_last_col_int { -1 }
1127   }
1128   { \int_set:Nn \l_@@_last_col_int { #1 } } ,
1129   columns-type .tl_set:N = \l_@@_columns_type_tl ,
1130   columns-type .value_required:n = true ,
1131   l .meta:n = { columns-type = l } ,
1132   r .meta:n = { columns-type = r } ,
1133   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1134   delimiters / color .value_required:n = true ,

```

```

1135     delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1136     delimiters / max-width .default:n = true ,
1137     delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
1138     delimiters .value_required:n = true ,
1139     small .bool_set:N = \l_@@_small_bool ,
1140     small .value_forbidden:n = true ,
1141     unknown .code:n = \@@_error:n { Unknown~key~for~NiceMatrix }
1142 }

```

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

```

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

```

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

```

1145     small .bool_set:N = \l_@@_small_bool ,
1146     small .value_forbidden:n = true ,
1147     last-col .code:n = \tl_if_empty:nF { #1 }
1148         { \@@_error:n { last-col-non-empty-for~NiceArray } }
1149         \int_zero:N \l_@@_last_col_int ,
1150     r .code:n = \@@_error:n { r-or-l-with-preamble } ,
1151     l .code:n = \@@_error:n { r-or-l-with-preamble } ,
1152     unknown .code:n = \@@_error:n { Unknown~key~for~NiceArray }
1153 }
1154 \keys_define:nn { nicematrix / pNiceArray }
1155 {
1156     first-col .code:n = \int_zero:N \l_@@_first_col_int ,
1157     last-col .code:n = \tl_if_empty:nF { #1 }
1158         { \@@_error:n { last-col-non-empty-for~NiceArray } }
1159         \int_zero:N \l_@@_last_col_int ,
1160     first-row .code:n = \int_zero:N \l_@@_first_row_int ,
1161     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1162     delimiters / color .value_required:n = true ,
1163     delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1164     delimiters / max-width .default:n = true ,
1165     delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
1166     delimiters .value_required:n = true ,
1167     small .bool_set:N = \l_@@_small_bool ,
1168     small .value_forbidden:n = true ,
1169     r .code:n = \@@_error:n { r-or-l-with-preamble } ,
1170     l .code:n = \@@_error:n { r-or-l-with-preamble } ,
1171     unknown .code:n = \@@_error:n { Unknown~key~for~NiceMatrix }
1172 }

```

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

```

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

```

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

```

1175     width .code:n = \dim_set:Nn \l_@@_width_dim { #1 }
1176         \bool_set_true:N \l_@@_width_used_bool ,
1177     width .value_required:n = true ,
1178     notes .code:n = \keys_set:nn { nicematrix / notes } { #1 } ,
1179     tabularnote .tl_gset:N = \g_@@_tabularnote_tl ,
1180     tabularnote .value_required:n = true ,
1181     caption .tl_set:N = \l_@@_caption_tl ,
1182     caption .value_required:n = true ,
1183     short-caption .tl_set:N = \l_@@_short_caption_tl ,

```

```

1184     short-caption .value_required:n = true ,
1185     label .tl_set:N = \l_@@_label_tl ,
1186     label .value_required:n = true ,
1187     last-col .code:n = \tl_if_empty:nF {#1}
1188         { \@@_error:n { last-col-non-empty-for-NiceArray } }
1189         \int_zero:N \l_@@_last_col_int ,
1190     r .code:n = \@@_error:n { r-or-l-with-preamble } ,
1191     l .code:n = \@@_error:n { r-or-l-with-preamble } ,
1192     unknown .code:n = \@@_error:n { Unknown-key-for-NiceTabular }
1193 }

```

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

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

```

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

```

9 Important code used by {NiceArrayWithDelims}

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

```

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

```

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

```
1207     \tl_gclear:N \g_@@_cell_after_hook_tl
```

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

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

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

```
1209     \int_gincr:N \c@jCol
```

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

```

1210     \int_compare:nNnT \c@jCol = \c_one_int
1211     { \int_compare:nNnT \l_@@_first_col_int = \c_one_int \@@_begin_of_row: }

```

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

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

The following command is nullified in the tabulars.

```
1213     \@@_tuning_not_tabular_begin:
```

```

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

```

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

```

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

```

We will use a version a little more efficient.

```

1218 \cs_new_protected:Npn \@@_tuning_first_row:
1219 {
1220   \if_int_compare:w \c@iRow = \c_zero_int
1221     \if_int_compare:w \c@jCol > \c_zero_int
1222       \l_@@_code_for_first_row_tl
1223       \xglobal \colorlet { nicematrix-first-row } { . }
1224   \fi:
1225   \fi:
1226 }

```

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

```

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

```

We will use a version a little more efficient.

```

1227 \cs_new_protected:Npn \@@_tuning_last_row:
1228 {
1229   \if_int_compare:w \c@iRow = \l_@@_last_row_int
1230     \l_@@_code_for_last_row_tl
1231     \xglobal \colorlet { nicematrix-last-row } { . }
1232   \fi:
1233 }

```

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

```

1234 \cs_set_eq:NN \@@_tuning_key_small: \prg_do_nothing:

```

The following commands are nullified in the tabulars.

```

1235 \cs_set_nopar:Npn \@@_tuning_not_tabular_begin:
1236 {
1237   \c_math_toggle_token

```

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

```

1238 \@@_tuning_key_small:
1239 }
1240 \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.

```

1241 \cs_new_protected:Npn \@@_begin_of_row:
1242 {
1243   \int_gincr:N \c@iRow
1244   \dim_gset_eq:NN \g_@@_dp_ante_last_row_dim \g_@@_dp_last_row_dim
1245   \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
1246   \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1247   \pgfpicture
1248   \pgfrememberpicturepositiononpagetrue
1249   \pgfcoordinate
1250   { \@@_env: - row - \int_use:N \c@iRow - base }
1251   { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
1252   \str_if_empty:NF \l_@@_name_str
1253   {
1254     \pgfnodealias
1255     { \l_@@_name_str - row - \int_use:N \c@iRow - base }
1256     { \@@_env: - row - \int_use:N \c@iRow - base }
1257   }
1258   \endpgfpicture
1259 }

```

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.

```

1260 \cs_new_protected:Npn \@@_update_for_first_and_last_row:
1261 {
1262   \int_if_zero:nTF \c@iRow
1263   {
1264     \dim_gset:Nn \g_@@_dp_row_zero_dim
1265     { \dim_max:nn \g_@@_dp_row_zero_dim { \box_dp:N \l_@@_cell_box } }
1266     \dim_gset:Nn \g_@@_ht_row_zero_dim
1267     { \dim_max:nn \g_@@_ht_row_zero_dim { \box_ht:N \l_@@_cell_box } }
1268   }
1269   {
1270     \int_compare:nNnT \c@iRow = \c_one_int
1271     {
1272       \dim_gset:Nn \g_@@_ht_row_one_dim
1273       { \dim_max:nn \g_@@_ht_row_one_dim { \box_ht:N \l_@@_cell_box } }
1274     }
1275   }
1276 }
1277 \cs_new_protected:Npn \@@_rotate_cell_box:
1278 {
1279   \box_rotate:Nn \l_@@_cell_box { 90 }
1280   \bool_if:NTF \g_@@_rotate_c_bool
1281   {
1282     \hbox_set:Nn \l_@@_cell_box
1283     {
1284       \c_math_toggle_token
1285       \vcenter { \box_use:N \l_@@_cell_box }
1286       \c_math_toggle_token
1287     }
1288   }
1289   {
1290     \int_compare:nNnT \c@iRow = \l_@@_last_row_int
1291     {
1292       \vbox_set_top:Nn \l_@@_cell_box
1293       {

```

```

1294         \vbox_to_zero:n { }
1295         \skip_vertical:n { - \box_ht:N \@arstrutbox + 0.8 ex }
1296         \box_use:N \l_@@_cell_box
1297     }
1298 }
1299 }
1300 \bool_gset_false:N \g_@@_rotate_bool
1301 \bool_gset_false:N \g_@@_rotate_c_bool
1302 }
1303 \cs_new_protected:Npn \@@_adjust_size_box:
1304 {
1305     \dim_compare:nNnT \g_@@_blocks_wd_dim > \c_zero_dim
1306     {
1307         \box_set_wd:Nn \l_@@_cell_box
1308         { \dim_max:nn { \box_wd:N \l_@@_cell_box } \g_@@_blocks_wd_dim }
1309         \dim_gzero:N \g_@@_blocks_wd_dim
1310     }
1311     \dim_compare:nNnT \g_@@_blocks_dp_dim > \c_zero_dim
1312     {
1313         \box_set_dp:Nn \l_@@_cell_box
1314         { \dim_max:nn { \box_dp:N \l_@@_cell_box } \g_@@_blocks_dp_dim }
1315         \dim_gzero:N \g_@@_blocks_dp_dim
1316     }
1317     \dim_compare:nNnT \g_@@_blocks_ht_dim > \c_zero_dim
1318     {
1319         \box_set_ht:Nn \l_@@_cell_box
1320         { \dim_max:nn { \box_ht:N \l_@@_cell_box } \g_@@_blocks_ht_dim }
1321         \dim_gzero:N \g_@@_blocks_ht_dim
1322     }
1323 }
1324 \cs_new_protected:Npn \@@_cell_end:
1325 {

```

The following command is nullified in the tabulars.

```

1326     \@@_tuning_not_tabular_end:
1327     \hbox_set_end:
1328     \@@_cell_end_i:
1329 }
1330 \cs_new_protected:Npn \@@_cell_end_i:
1331 {

```

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.

```

1332     \g_@@_cell_after_hook_tl
1333     \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
1334     \@@_adjust_size_box:
1335     \box_set_ht:Nn \l_@@_cell_box
1336     { \box_ht:N \l_@@_cell_box + \l_@@_cell_space_top_limit_dim }
1337     \box_set_dp:Nn \l_@@_cell_box
1338     { \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”).

```

1339     \@@_update_max_cell_width:

```

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

```

1340     \@@_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.

```

1341 \bool_if:NTF \g_@@_empty_cell_bool
1342 { \box_use_drop:N \l_@@_cell_box }
1343 {
1344   \bool_if:NTF \g_@@_not_empty_cell_bool
1345   \@@_node_for_cell:
1346   {
1347     \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > \c_zero_dim
1348     \@@_node_for_cell:
1349     { \box_use_drop:N \l_@@_cell_box }
1350   }
1351 }
1352 \int_compare:nNnT \c@jCol > \g_@@_col_total_int
1353 { \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
1354 \bool_gset_false:N \g_@@_empty_cell_bool
1355 \bool_gset_false:N \g_@@_not_empty_cell_bool
1356 }

```

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

```

1357 \cs_new_protected:Npn \@@_update_max_cell_width:
1358 {
1359   \dim_gset:Nn \g_@@_max_cell_width_dim
1360   { \dim_max:nn \g_@@_max_cell_width_dim { \box_wd:N \l_@@_cell_box } }
1361 }

```

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`).

```

1362 \cs_new_protected:Npn \@@_cell_end_for_w_s:
1363 {
1364   \@@_math_toggle:
1365   \hbox_set_end:
1366   \bool_if:NF \g_@@_rotate_bool
1367   {
1368     \hbox_set:Nn \l_@@_cell_box
1369     {
1370       \makebox [ \l_@@_col_width_dim ] [ s ]
1371       { \hbox_unpack_drop:N \l_@@_cell_box }
1372     }
1373   }
1374   \@@_cell_end_i:
1375 }

```

```

1376 \pgfset
1377 {
1378   nicematrix / cell-node /.style =
1379   {
1380     inner-sep = \c_zero_dim ,

```



```

1381     minimum~width = \c_zero_dim
1382   }
1383 }

```

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

```

1384 \cs_new_protected:Npn \@@_node_for_cell:
1385 {
1386   \pgfpicture
1387   \pgfsetbaseline \c_zero_dim
1388   \pgfrememberpicturepositiononpagetrue
1389   \pgfset { nicematrix / cell-node }
1390   \pgfnode
1391     { rectangle }
1392     { base }
1393   {

```

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

```

1394     \set@color
1395     \box_use_drop:N \l_@@_cell_box
1396   }
1397   { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1398   { \l_@@_pgf_node_code_tl }
1399   \str_if_empty:NF \l_@@_name_str
1400   {
1401     \pgfnodealias
1402       { \l_@@_name_str - \int_use:N \c@iRow - \int_use:N \c@jCol }
1403       { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1404   }
1405   \endpgfpicture
1406 }

```

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.

```

1407 \cs_new_protected:Npn \@@_patch_node_for_cell:n #1
1408 {
1409   \cs_new_protected:Npn \@@_patch_node_for_cell:
1410   {
1411     \hbox_set:Nn \l_@@_cell_box
1412     {
1413       \box_move_up:nn { \box_ht:N \l_@@_cell_box }
1414       \hbox_overlap_left:n
1415       {
1416         \pgfsys@markposition
1417         { \@@_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.

```

1418         #1
1419       }
1420       \box_use:N \l_@@_cell_box
1421       \box_move_down:nn { \box_dp:N \l_@@_cell_box }
1422       \hbox_overlap_left:n
1423       {
1424         \pgfsys@markposition
1425         { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - SE }
1426         #1
1427       }
1428     }
1429   }
1430 }

```

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

```

1431 \bool_lazy_or:nnTF \sys_if_engine_xetex_p: \sys_if_output_dvi_p:
1432 {
1433   \@@_patch_node_for_cell:n
1434     { \skip_horizontal:n { 0.5 \box_wd:N \l_@@_cell_box } }
1435 }
1436 { \@@_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}

```

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

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).

```

1437 \cs_new_protected:Npn \@@_instruction_of_type:nnn #1 #2 #3
1438 {
1439   \bool_if:nTF { #1 } \tl_gput_left:ce \tl_gput_right:ce
1440     { g_@@_#2 _ lines _ tl }
1441   {
1442     \use:c { @@ _ draw _ #2 : nnn }
1443     { \int_use:N \c@iRow }
1444     { \int_use:N \c@jCol }
1445     { \exp_not:n { #3 } }
1446   }
1447 }

1448 \cs_generate_variant:Nn \@@_array:n { o }
1449 \cs_new_protected:Npn \@@_array:n
1450 {
1451   % \begin{macrocode}
1452   \dim_set:Nn \col@sep
1453     { \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
1454   \dim_compare:nNnTF \l_@@_tabular_width_dim = \c_zero_dim
1455     { \cs_set_nopar:Npn \@halignto { } }
1456     { \cs_set_nopar:Npe \@halignto { to \dim_use:N \l_@@_tabular_width_dim } }

1457   \@tabarray
1458   \l_@@_baseline_tl may have the value t, c or b. However, if the value is b, we compose the
1459   \array (of array) with the option t and the right translation will be done further. Remark that
1460   \str_if_eq:onTF is fully expandable and we need something fully expandable here.
1461   [ \str_if_eq:onTF \l_@@_baseline_tl c c t ]
1462 }

```

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`.

```

1460 \bool_if:NTF \c_@@_tagging_array_bool
1461 { \cs_set_eq:NN \@@_old_ar@ialign: \ar@ialign }
1462 { \cs_set_eq:NN \@@_old_ialign: \ialign }

```

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

```

1463 \cs_new_protected:Npn \@@_create_row_node:
1464 {
1465   \int_compare:nNnT \c@iRow > \g_@@_last_row_node_int
1466   {
1467     \int_gset_eq:NN \g_@@_last_row_node_int \c@iRow
1468     \@@_create_row_node_i:
1469   }
1470 }
1471 \cs_new_protected:Npn \@@_create_row_node_i:
1472 {

```

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

```

1473   \hbox
1474   {
1475     \bool_if:NT \l_@@_code_before_bool
1476     {
1477       \vtop
1478       {
1479         \skip_vertical:N 0.5\arrayrulewidth
1480         \pgfsys@markposition
1481         { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1482         \skip_vertical:N -0.5\arrayrulewidth
1483       }
1484     }
1485     \pgfpicture
1486     \pgfrememberpicturepositiononpagetrue
1487     \pgfcoordinate { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1488     { \pgfpoint \c_zero_dim { - 0.5 \arrayrulewidth } }
1489     \str_if_empty:NF \l_@@_name_str
1490     {
1491       \pgfnodealias
1492       { \l_@@_name_str - row - \int_eval:n { \c@iRow + 1 } }
1493       { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1494     }
1495     \endpgfpicture
1496   }
1497 }

```

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

```

1498 \cs_new:Npn \@@_everycr: { \noalign { \@@_everycr_i: } }
1499 \cs_new_protected:Npn \@@_everycr_i:
1500 {
1501   \bool_if:NT \c_@@_testphase_table_bool
1502   {
1503     \tbl_if_row_was_started:T { \UseTaggingSocket { tbl / row / end } }
1504     \tbl_update_cell_data_for_next_row:
1505   }
1506   \int_gzero:N \c@jCol
1507   \bool_gset_false:N \g_@@_after_col_zero_bool
1508   \bool_if:NF \g_@@_row_of_col_done_bool
1509   {
1510     \@@_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).

```

1511 \tl_if_empty:NF \l_@@_hlines_clist
1512 {
1513   \tl_if_eq:NNF \l_@@_hlines_clist \c_@@_all_tl
1514   {
1515     \clist_if_in:NeT
1516       \l_@@_hlines_clist
1517       { \int_eval:n { \c@iRow + 1 } }
1518   }
1519 }

```

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.

```

1520 \int_compare:nNnT \c@iRow > { -1 }
1521 {
1522   \int_compare:nNnF \c@iRow = \l_@@_last_row_int
1523     { \hrule height \arrayrulewidth width \c_zero_dim }
1524 }
1525 }
1526 }
1527 }
1528 }

```

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

```

1529 \cs_set_protected:Npn \@@_renew_dots:
1530 {
1531   \cs_set_eq:NN \ldots \@@_Ldots
1532   \cs_set_eq:NN \cdots \@@_Cdots
1533   \cs_set_eq:NN \vdots \@@_Vdots
1534   \cs_set_eq:NN \ddots \@@_Ddots
1535   \cs_set_eq:NN \iddots \@@_Iddots
1536   \cs_set_eq:NN \dots \@@_Ldots
1537   \cs_set_eq:NN \hdotsfor \@@_Hdotsfor:
1538 }
1539 \cs_new_protected:Npn \@@_test_color_inside:
1540 {
1541   \bool_if:NF \l_@@_color_inside_bool
1542   {

```

We will issue an error only during the first run.

```

1543     \bool_if:NF \g_@@_aux_found_bool
1544     { \@@_error:n { without~color~inside } }
1545   }
1546 }

```

```

1547 \cs_new_protected:Npn \@@_redefine_everycr:
1548 { \everycr { \@@_everycr: } }
1549 \hook_gput_code:nnn { begindocument } { . }
1550 {
1551   \IfPackageLoadedT { colortbl }
1552   {
1553     \cs_set_protected:Npn \@@_redefine_everycr:
1554     {
1555       \CT@everycr
1556       {
1557         \noalign { \cs_gset_eq:NN \CT@row@color \prg_do_nothing: }
1558         \@@_everycr:
1559       }
1560     }
1561   }
1562 }

```

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 redefinition⁴.

```

1563 \hook_gput_code:nnn { begindocument } { . }
1564 {
1565   \IfPackageLoadedTF { booktabs }
1566   {
1567     \cs_new_protected:Npn \@@_patch_booktabs:
1568     { \tl_put_left:Nn \@BTnormal \@@_create_row_node_i: }
1569   }
1570   { \cs_new_protected:Npn \@@_patch_booktabs: { } }
1571 }

```

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

```

1572 \cs_new_protected:Npn \@@_some_initialization:
1573 {
1574   \dim_gset:Nn \g_@@_dp_row_zero_dim { \box_dp:N \@arstrutbox }
1575   \dim_gset:Nn \g_@@_ht_row_zero_dim { \box_ht:N \@arstrutbox }
1576   \dim_gset_eq:NN \g_@@_ht_row_one_dim \g_@@_ht_row_zero_dim
1577   \dim_gzero:N \g_@@_dp_ante_last_row_dim
1578   \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1579   \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
1580 }

```

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

```

1581 \cs_new_protected:Npn \@@_pre_array_ii:
1582 {

```

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

```

1583   \int_gzero:N \g_@@_total_X_weight_int
1584   \@@_expand_clist:N \l_@@_hlines_clist
1585   \@@_expand_clist:N \l_@@_vlines_clist
1586   \@@_patch_booktabs:
1587   \box_clear_new:N \l_@@_cell_box
1588   \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}`).

```

1589   \bool_if:NT \l_@@_small_bool
1590   {
1591     \cs_set_nopar:Npn \arraystretch { 0.47 }
1592     \dim_set:Nn \arraycolsep { 1.45 pt }

```

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

```

1593     \cs_set_eq:NN \@@_tuning_key_small: \scriptstyle
1594   }

```

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}`.

```

1595 \bool_if:NT \g_@@_recreate_cell_nodes_bool
1596 {
1597   \tl_put_right:Nn \@@_begin_of_row:
1598   {
1599     \pgfsys@markposition
1600     { \@@_env: - row - \int_use:N \c@iRow - base }
1601   }
1602 }

```

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`.

```

1603 \bool_if:NTF \c_@@_tagging_array_bool
1604 {
1605   \cs_set_nopar:Npn \ar@ialign
1606   {
1607     \bool_if:NT \c_@@_testphase_table_bool \tbl_init_cell_data_for_table:
1608     \@@_redefine_everycr:
1609     \dim_zero:N \tabskip
1610     \@@_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`.

```

1611     \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign:
1612     \halign
1613   }
1614 }

```

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).

```

1615 {
1616   \cs_set_nopar:Npn \ialign
1617   {
1618     \@@_redefine_everycr:
1619     \dim_zero:N \tabskip
1620     \@@_some_initialization:
1621     \cs_set_eq:NN \ialign \@@_old_ialign:
1622     \halign
1623   }
1624 }

```

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).

```

1625 \cs_set_eq:NN \@@_old_ldots \ldots
1626 \cs_set_eq:NN \@@_old_cdots \cdots
1627 \cs_set_eq:NN \@@_old_vdots \vdots
1628 \cs_set_eq:NN \@@_old_ddots \ddots
1629 \cs_set_eq:NN \@@_old_iddots \iddots
1630 \bool_if:NTF \l_@@_standard_cline_bool
1631 { \cs_set_eq:NN \cline \@@_standard_cline }
1632 { \cs_set_eq:NN \cline \@@_cline }
1633 \cs_set_eq:NN \Ldots \@@_Ldots
1634 \cs_set_eq:NN \Cdots \@@_Cdots
1635 \cs_set_eq:NN \Vdots \@@_Vdots
1636 \cs_set_eq:NN \Ddots \@@_Ddots
1637 \cs_set_eq:NN \Iddots \@@_Iddots
1638 \cs_set_eq:NN \Hline \@@_Hline:
1639 \cs_set_eq:NN \Hspace \@@_Hspace:
1640 \cs_set_eq:NN \Hdotsfor \@@_Hdotsfor:
1641 \cs_set_eq:NN \Vdotsfor \@@_Vdotsfor:
1642 \cs_set_eq:NN \Block \@@_Block:
1643 \cs_set_eq:NN \rotate \@@_rotate:

```

```

1644 \cs_set_eq:NN \OnlyMainNiceMatrix \@@_OnlyMainNiceMatrix:n
1645 \cs_set_eq:NN \dotfill \@@_dotfill:
1646 \cs_set_eq:NN \CodeAfter \@@_CodeAfter:
1647 \cs_set_eq:NN \diagbox \@@_diagbox:nn
1648 \cs_set_eq:NN \NotEmpty \@@_NotEmpty:
1649 \cs_set_eq:NN \RowStyle \@@_RowStyle:n
1650 \seq_map_inline:Nn \l_@@_custom_line_commands_seq
1651 { \cs_set_eq:cc { ##1 } { nicematrix - ##1 } }
1652 \cs_set_eq:NN \cellcolor \@@_cellcolor_tabular
1653 \cs_set_eq:NN \rowcolor \@@_rowcolor_tabular
1654 \cs_set_eq:NN \rowcolors \@@_rowcolors_tabular
1655 \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors_tabular
1656 \int_compare:nNt \l_@@_first_row_int > \c_zero_int
1657 { \cs_set_eq:NN \@@_tuning_first_row: \prg_do_nothing: }
1658 \int_compare:nNt \l_@@_last_row_int < \c_zero_int
1659 { \cs_set_eq:NN \@@_tuning_last_row: \prg_do_nothing: }
1660 \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:`.

```

1661 \cs_set_eq:NN \multicolumn \@@_multicolumn:nnn
1662 \hook_gput_code:nnn { env / tabular / begin } { nicematrix }
1663 { \cs_set_eq:NN \multicolumn \@@_old_multicolumn }
1664 \@@_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!).

```

1665 \tl_if_exist:NT \l_@@_note_in_caption_tl
1666 {
1667   \tl_if_empty:NF \l_@@_note_in_caption_tl
1668   {
1669     \int_gset_eq:NN \g_@@_notes_caption_int \l_@@_note_in_caption_tl
1670     \int_gset:Nn \c@tabularnote { \l_@@_note_in_caption_tl }
1671   }
1672 }

```

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).

```

1673 \seq_gclear:N \g_@@_multicolumn_cells_seq
1674 \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).

```

1675 \int_gset:Nn \c@iRow { \l_@@_first_row_int - 1 }

```

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

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

```

1676 \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.

```

1677 \int_gzero_new:N \g_@@_col_total_int
1678 \cs_set_eq:NN \@ifnextchar \new@ifnextchar
1679 \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.

```

1680 \tl_gclear_new:N \g_@@_Cdots_lines_tl
1681 \tl_gclear_new:N \g_@@_Ldots_lines_tl
1682 \tl_gclear_new:N \g_@@_Vdots_lines_tl
1683 \tl_gclear_new:N \g_@@_Ddots_lines_tl
1684 \tl_gclear_new:N \g_@@_Iddots_lines_tl
1685 \tl_gclear_new:N \g_@@_HVDotsfor_lines_tl

1686 \tl_gclear:N \g_nicematrix_code_before_tl
1687 \tl_gclear:N \g_@@_pre_code_before_tl
1688 }

```

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

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

```

1689 \cs_new_protected:Npn \@@_pre_array:
1690 {
1691   \cs_if_exist:NT \theiRow { \int_set_eq:NN \l_@@_old_iRow_int \c@iRow }
1692   \int_gzero_new:N \c@iRow
1693   \cs_if_exist:NT \thejCol { \int_set_eq:NN \l_@@_old_jCol_int \c@jCol }
1694   \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).

```

1695 \int_compare:nNnT \l_@@_last_row_int = { -1 }
1696 {
1697   \bool_set_true:N \l_@@_last_row_without_value_bool
1698   \bool_if:NT \g_@@_aux_found_bool
1699     { \int_set:Nn \l_@@_last_row_int { \seq_item:Nn \g_@@_size_seq 3 } }
1700 }
1701 \int_compare:nNnT \l_@@_last_col_int = { -1 }
1702 {
1703   \bool_if:NT \g_@@_aux_found_bool
1704     { \int_set:Nn \l_@@_last_col_int { \seq_item:Nn \g_@@_size_seq 6 } }
1705 }

```

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”.

```

1706 \int_compare:nNnT \l_@@_last_row_int > { -2 }
1707 {
1708   \tl_put_right:Nn \@@_update_for_first_and_last_row:
1709     {
1710       \dim_gset:Nn \g_@@_ht_last_row_dim
1711         { \dim_max:nn \g_@@_ht_last_row_dim { \box_ht:N \l_@@_cell_box } }
1712       \dim_gset:Nn \g_@@_dp_last_row_dim
1713         { \dim_max:nn \g_@@_dp_last_row_dim { \box_dp:N \l_@@_cell_box } }
1714     }
1715 }

1716 \seq_gclear:N \g_@@_cols_vlism_seq
1717 \seq_gclear:N \g_@@_submatrix_seq

```

Now the `\CodeBefore`.

```

1718 \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.

```
1719 \seq_gclear:N \g_@@_pos_of_blocks_seq
```

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

```
1720 \seq_gclear_new:N \g_@@_multicolumn_cells_seq
```

```
1721 \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.

```
1722 \int_gset:Nn \g_@@_last_row_node_int { -2 }
```

The value `-2` is important.

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

```
1723 \@@_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.

```
1724 \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]`).

```
1725 \dim_zero_new:N \l_@@_left_delim_dim
```

```
1726 \dim_zero_new:N \l_@@_right_delim_dim
```

```
1727 \bool_if:NTF \g_@@_delims_bool
```

```
1728 {
```

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

```
1729 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_left_delim_tl $ }
```

```
1730 \dim_set:Nn \l_@@_left_delim_dim { \box_wd:N \l_tmpa_box }
```

```
1731 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_right_delim_tl $ }
```

```
1732 \dim_set:Nn \l_@@_right_delim_dim { \box_wd:N \l_tmpa_box }
```

```
1733 }
```

```
1734 {
```

```
1735 \dim_gset:Nn \l_@@_left_delim_dim
```

```
1736 { 2 \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
```

```
1737 \dim_gset_eq:NN \l_@@_right_delim_dim \l_@@_left_delim_dim
```

```
1738 }
```

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).

```
1739 \hbox_set:Nw \l_@@_the_array_box
```

```
1740 \bool_if:NT \c_@@_testphase_table_bool
```

```
1741 { \UseTaggingSocket { tbl / hmode / begin } }
```

```
1742 \skip_horizontal:N \l_@@_left_margin_dim
```

```
1743 \skip_horizontal:N \l_@@_extra_left_margin_dim
```

```
1744 \c_math_toggle_token
```

```
1745 \bool_if:NTF \l_@@_light_syntax_bool
```

```
1746 { \use:c { @@-light-syntax } }
```

```
1747 { \use:c { @@-normal-syntax } }
```

```
1748 }
```

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

```

1749 \cs_new_protected_nopar:Npn \@@_CodeBefore_Body:w #1 \Body
1750 {
1751   \tl_set:Nn \l_tmpa_tl { #1 }
1752   \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
1753     { \@@_rescan_for_spanish:N \l_tmpa_tl }
1754   \tl_gput_left:No \g_@@_pre_code_before_tl \l_tmpa_tl
1755   \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...

```

1756   \@@_pre_array:
1757 }

```

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).

```

1758 \cs_new_protected:Npn \@@_pre_code_before:
1759 {

```

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).

```

1760   \int_set:Nn \c{iRow} { \seq_item:Nn \g_@@_size_seq 2 }
1761   \int_set:Nn \c{jCol} { \seq_item:Nn \g_@@_size_seq 5 }
1762   \int_set_eq:NN \g_@@_row_total_int { \seq_item:Nn \g_@@_size_seq 3 }
1763   \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.

```

1764   \pgfsys@markposition { \@@_env: - position }
1765   \pgfsys@getposition { \@@_env: - position } \@@_picture_position:
1766   \pgfpicture
1767   \pgf@relevantforpicturesizefalse

```

First, the recreation of the `row` nodes.

```

1768   \int_step_inline:nnn \l_@@_first_row_int { \g_@@_row_total_int + 1 }
1769   {
1770     \pgfsys@getposition { \@@_env: - row - ##1 } \@@_node_position:
1771     \pgfcoordinate { \@@_env: - row - ##1 }
1772     { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1773   }

```

Now, the recreation of the `col` nodes.

```

1774   \int_step_inline:nnn \l_@@_first_col_int { \g_@@_col_total_int + 1 }
1775   {
1776     \pgfsys@getposition { \@@_env: - col - ##1 } \@@_node_position:
1777     \pgfcoordinate { \@@_env: - col - ##1 }
1778     { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1779   }

```

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

```

1780   \@@_create_diag_nodes:

```

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

```
1781 \bool_if:NT \g_@@_recreate_cell_nodes_bool \@@_recreate_cell_nodes:
1782 \endpgfpicture
```

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

```
1783 \@@_create_blocks_nodes:
1784 \IfPackageLoadedT { tikz }
1785 {
1786   \tikzset
1787   {
1788     every-picture / .style =
1789     { overlay , name-prefix = \@@_env: - }
1790   }
1791 }
1792 \cs_set_eq:NN \cellcolor \@@_cellcolor
1793 \cs_set_eq:NN \rectanglecolor \@@_rectanglecolor
1794 \cs_set_eq:NN \roundedrectanglecolor \@@_roundedrectanglecolor
1795 \cs_set_eq:NN \rowcolor \@@_rowcolor
1796 \cs_set_eq:NN \rowcolors \@@_rowcolors
1797 \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors
1798 \cs_set_eq:NN \arraycolor \@@_arraycolor
1799 \cs_set_eq:NN \columncolor \@@_columncolor
1800 \cs_set_eq:NN \chessboardcolors \@@_chessboardcolors
1801 \cs_set_eq:NN \SubMatrix \@@_SubMatrix_in_code_before
1802 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
1803 \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
1804 }
```

```
1805 \cs_new_protected:Npn \@@_exec_code_before:
1806 {
1807   \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`.

```
1808 \@@_add_to_colors_seq:nn { { nocolor } } { }
1809 \bool_gset_false:N \g_@@_recreate_cell_nodes_bool
1810 \group_begin:
```

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

```
1811 \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`).

```
1812 \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
1813 { \@@_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`.

```
1814 \exp_last_unbraced:No \@@_CodeBefore_keys:
1815 \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.

```

1816     \@@_actually_color:
1817     \l_@@_code_before_tl
1818     \q_stop
1819     \bool_if:NT \l_@@_tabular_bool \c_math_toggle_token
1820     \group_end:
1821     \bool_if:NT \g_@@_recreate_cell_nodes_bool
1822     { \tl_put_left:Nn \@@_node_for_cell: \@@_patch_node_for_cell: }
1823 }

1824 \keys_define:nn { nicematrix / CodeBefore }
1825 {
1826     create-cell-nodes .bool_gset:N = \g_@@_recreate_cell_nodes_bool ,
1827     create-cell-nodes .default:n = true ,
1828     sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1829     sub-matrix .value_required:n = true ,
1830     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1831     delimiters / color .value_required:n = true ,
1832     unknown .code:n = \@@_error:n { Unknown~key~for~CodeBefore }
1833 }

1834 \NewDocumentCommand \@@_CodeBefore_keys: { 0 { } }
1835 {
1836     \keys_set:nn { nicematrix / CodeBefore } { #1 }
1837     \@@_CodeBefore:w
1838 }

```

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.

```

1839 \cs_new_protected:Npn \@@_CodeBefore:w #1 \q_stop
1840 {
1841     \bool_if:NT \g_@@_aux_found_bool
1842     {
1843         \@@_pre_code_before:
1844         #1
1845     }
1846 }

```

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.

```

1847 \cs_new_protected:Npn \@@_recreate_cell_nodes:
1848 {
1849     \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
1850     {
1851         \pgfsys@getposition { \@@_env: - ##1 - base } \@@_node_position:
1852         \pgfcoordinate { \@@_env: - row - ##1 - base }
1853         { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1854         \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
1855         {
1856             \cs_if_exist:cT
1857             { pgf @ sys @ pdf @ mark @ pos @ \@@_env: - ##1 - #####1 - NW }
1858             {
1859                 \pgfsys@getposition
1860                 { \@@_env: - ##1 - #####1 - NW }
1861                 \@@_node_position:
1862                 \pgfsys@getposition
1863                 { \@@_env: - ##1 - #####1 - SE }

```

```

1864         \@@_node_position_i:
1865         \@@_pgf_rect_node:nnn
1866         { \@@_env: - ##1 - #####1 }
1867         { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1868         { \pgfpointdiff \@@_picture_position: \@@_node_position_i: }
1869     }
1870 }
1871 }
1872 \int_step_inline:nn \c@iRow
1873 {
1874     \pgfnodealias
1875     { \@@_env: - ##1 - last }
1876     { \@@_env: - ##1 - \int_use:N \c@jCol }
1877 }
1878 \int_step_inline:nn \c@jCol
1879 {
1880     \pgfnodealias
1881     { \@@_env: - last - ##1 }
1882     { \@@_env: - \int_use:N \c@iRow - ##1 }
1883 }
1884 \@@_create_extra_nodes:
1885 }

```

```

1886 \cs_new_protected:Npn \@@_create_blocks_nodes:
1887 {
1888     \pgfpicture
1889     \pgf@relevantforpicturesizefalse
1890     \pgfrememberpicturepositiononpagetrue
1891     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
1892     { \@@_create_one_block_node:nnnnn ##1 }
1893     \endpgfpicture
1894 }

```

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

```

1895 \cs_new_protected:Npn \@@_create_one_block_node:nnnnn #1 #2 #3 #4 #5
1896 {
1897     \tl_if_empty:nF { #5 }
1898     {
1899         \@@_qpoint:n { col - #2 }
1900         \dim_set_eq:NN \l_tmpa_dim \pgf@x
1901         \@@_qpoint:n { #1 }
1902         \dim_set_eq:NN \l_tmpb_dim \pgf@y
1903         \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
1904         \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
1905         \@@_qpoint:n { \int_eval:n { #3 + 1 } }
1906         \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
1907         \@@_pgf_rect_node:nnnnn
1908         { \@@_env: - #5 }
1909         { \dim_use:N \l_tmpa_dim }
1910         { \dim_use:N \l_tmpb_dim }
1911         { \dim_use:N \l_@@_tmpc_dim }
1912         { \dim_use:N \l_@@_tmpd_dim }
1913     }
1914 }

```

```

1915 \cs_new_protected:Npn \@@_patch_for_revtext:
1916 {

```

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).

```

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

```

11 The environment `{NiceArrayWithDelims}`

```

1931 \NewDocumentEnvironment { NiceArrayWithDelims }
1932 { m m O { } m ! O { } t \CodeBefore }
1933 {
1934 \bool_if:NT \c_@@_revtex_bool \@_patch_for_revtex:
1935 \@_provide_pgfsyspdfmark:
1936 \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.

```

1937 \bgroup

1938 \tl_gset:Nn \g_@@_left_delim_tl { #1 }
1939 \tl_gset:Nn \g_@@_right_delim_tl { #2 }
1940 \tl_gset:Nn \g_@@_user_preamble_tl { #4 }
1941 \tl_if_empty:NT \g_@@_user_preamble_tl { \@@_fatal:n { empty-preamble } }

1942 \int_gzero:N \g_@@_block_box_int
1943 \dim_zero:N \g_@@_width_last_col_dim
1944 \dim_zero:N \g_@@_width_first_col_dim
1945 \bool_gset_false:N \g_@@_row_of_col_done_bool
1946 \str_if_empty:NT \g_@@_name_env_str
1947 { \str_gset:Nn \g_@@_name_env_str { NiceArrayWithDelims } }
1948 \bool_if:NTF \l_@@_tabular_bool
1949 \mode_leave_vertical:
1950 \@_test_if_math_mode:
1951 \bool_if:NT \l_@@_in_env_bool { \@@_fatal:n { Yet~in~env } }
1952 \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.

```

1953 \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.

```

1954 \cs_if_exist:NT \tikz@library@external@loaded

```

⁷e.g. `\color[rgb]{0.5,0.5,0}`

```

1955 {
1956   \tikzexternaldisable
1957   \cs_if_exist:NT \ifstandalone
1958   { \tikzset { external / optimize = false } }
1959 }

```

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

```

1960 \int_gincr:N \g_@@_env_int
1961 \bool_if:NF \l_@@_block_auto_columns_width_bool
1962 { \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`).

```

1963 \seq_gclear:N \g_@@_blocks_seq
1964 \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`.

```

1965 \seq_gclear:N \g_@@_pos_of_stroken_blocks_seq
1966 \seq_gclear:N \g_@@_pos_of_xdots_seq
1967 \tl_gclear_new:N \g_@@_code_before_tl
1968 \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.

```

1969 \tl_if_exist:cTF { c_@@ _ \int_use:N \g_@@_env_int _ tl }
1970 {
1971   \bool_gset_true:N \g_@@_aux_found_bool
1972   \use:c { c_@@ _ \int_use:N \g_@@_env_int _ tl }
1973 }
1974 { \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.

```

1975 \tl_gclear:N \g_@@_aux_tl
1976 \tl_if_empty:NF \g_@@_code_before_tl
1977 {
1978   \bool_set_true:N \l_@@_code_before_bool
1979   \tl_put_right:No \l_@@_code_before_tl \g_@@_code_before_tl
1980 }
1981 \tl_if_empty:NF \g_@@_pre_code_before_tl
1982 { \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`.

```

1983 \bool_if:NTF \g_@@_delims_bool
1984 { \keys_set:nn { nicematrix / pNiceArray } }
1985 { \keys_set:nn { nicematrix / NiceArray } }
1986 { #3 , #5 }

```

```

1987 \@@_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:`.

```

1988 \bool_if:nTF { #6 } \@@_CodeBefore_Body:w \@@_pre_array:
1989 }

```

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

```

1990 {

```

```

1991 \bool_if:NTF \l_@@_light_syntax_bool
1992 { \use:c { end @@-light-syntax } }
1993 { \use:c { end @@-normal-syntax } }
1994 \c_math_toggle_token
1995 \skip_horizontal:N \l_@@_right_margin_dim
1996 \skip_horizontal:N \l_@@_extra_right_margin_dim
1997
1998 % awful workaround
1999 \int_compare:nNtT \g_@@_col_total_int = \c_one_int
2000 {
2001   \dim_compare:nNtT \l_@@_columns_width_dim > \c_zero_dim
2002   {
2003     \skip_horizontal:N - \l_@@_columns_width_dim
2004     \bool_if:NTF \l_@@_tabular_bool
2005       { \skip_horizontal:n { - 2 \tabcolsep } }
2006       { \skip_horizontal:n { - 2 \arraycolsep } }
2007   }
2008 }
2009 \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.

```

2010 \bool_if:NT \l_@@_width_used_bool
2011 {
2012   \int_if_zero:nT \g_@@_total_X_weight_int
2013   { \@@_error_or_warning:n { width~without~X~columns } }
2014 }

```

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 .

```

2015 \int_compare:nNtT \g_@@_total_X_weight_int > \c_zero_int
2016 {
2017   \tl_gput_right:Ne \g_@@_aux_tl
2018   {
2019     \bool_set_true:N \l_@@_X_columns_aux_bool
2020     \dim_set:Nn \l_@@_X_columns_dim
2021     {
2022       \dim_compare:nNtTF
2023       {
2024         \dim_abs:n
2025         { \l_@@_width_dim - \box_wd:N \l_@@_the_array_box }
2026       }
2027       <
2028       { 0.001 pt }
2029       { \dim_use:N \l_@@_X_columns_dim }
2030       {
2031         \dim_eval:n
2032         {
2033           ( \l_@@_width_dim - \box_wd:N \l_@@_the_array_box )
2034           / \int_use:N \g_@@_total_X_weight_int
2035           + \l_@@_X_columns_dim
2036         }
2037       }
2038     }
2039   }
2040 }

```

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).

```

2041 \int_compare:nNtT \l_@@_last_row_int > { -2 }
2042 {

```



```

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

```

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

```

2052 \int_gset_eq:NN \c@jCol \g_@@_col_total_int
2053 \bool_if:NTF \g_@@_last_col_found_bool
2054 { \int_gdecr:N \c@jCol }
2055 {
2056   \int_compare:nNnT \l_@@_last_col_int > { -1 }
2057   { \@@_error:n { last-col-not-used } }
2058 }

```

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

```

2059 \int_gset_eq:NN \g_@@_row_total_int \c@iRow
2060 \int_compare:nNnT \l_@@_last_row_int > { -1 } { \int_gdecr:N \c@iRow }

```

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

```

2061 \int_if_zero:nT \l_@@_first_col_int
2062 { \skip_horizontal:N \g_@@_width_first_col_dim }

```

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

```

2063 \bool_if:nTF { ! \g_@@_delims_bool }
2064 {
2065   \tl_if_eq:NNTF \l_@@_baseline_tl \c_@@_c_tl
2066   \@@_use_arraybox_with_notes_c:
2067   {
2068     \tl_if_eq:NNTF \l_@@_baseline_tl \c_@@_b_tl
2069     \@@_use_arraybox_with_notes_b:
2070     \@@_use_arraybox_with_notes:
2071   }
2072 }

```

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).

```

2073 {
2074   \int_if_zero:nTF \l_@@_first_row_int
2075   {
2076     \dim_set_eq:NN \l_tmpa_dim \g_@@_dp_row_zero_dim
2077     \dim_add:Nn \l_tmpa_dim \g_@@_ht_row_zero_dim
2078   }
2079   { \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

```

2080 \int_compare:nNnTF \l_@@_last_row_int > { -2 }
2081 {
2082   \dim_set_eq:NN \l_tmpb_dim \g_@@_ht_last_row_dim
2083   \dim_add:Nn \l_tmpb_dim \g_@@_dp_last_row_dim
2084 }
2085 { \dim_zero:N \l_tmpb_dim }

```

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

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

```

2086 \hbox_set:Nn \l_tmpa_box
2087 {
2088   \c_math_toggle_token
2089   \@@_color:o \l_@@_delimiters_color_tl
2090   \exp_after:wN \left \g_@@_left_delim_tl
2091   \vcenter
2092   {

```

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.

```

2093   \skip_vertical:n { -\l_tmpa_dim - \arrayrulewidth }
2094   \hbox
2095   {
2096     \bool_if:NTF \l_@@_tabular_bool
2097     { \skip_horizontal:N -\tabcolsep }
2098     { \skip_horizontal:N -\arraycolsep }
2099     \@@_use_arraybox_with_notes_c:
2100     \bool_if:NTF \l_@@_tabular_bool
2101     { \skip_horizontal:N -\tabcolsep }
2102     { \skip_horizontal:N -\arraycolsep }
2103   }

```

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

```

2104   \skip_vertical:N -\l_tmpb_dim
2105   \skip_vertical:N \arrayrulewidth
2106 }
2107 \exp_after:wN \right \g_@@_right_delim_tl
2108 \c_math_toggle_token
2109 }

```

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.

```

2110 \bool_if:NTF \l_@@_delimiters_max_width_bool
2111 {
2112   \@@_put_box_in_flow_bis:nn
2113   \g_@@_left_delim_tl
2114   \g_@@_right_delim_tl
2115 }
2116 \@@_put_box_in_flow:
2117 }

```

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. ??).

```

2118 \bool_if:NT \g_@@_last_col_found_bool
2119 { \skip_horizontal:N \g_@@_width_last_col_dim }
2120 \bool_if:NT \l_@@_preamble_bool
2121 {
2122   \int_compare:nNnT \c_jCol < \g_@@_static_num_of_col_int
2123   { \@@_warning_gredirect_none:n { columns-not-used } }
2124 }
2125 \@@_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.

```

2126 \egroup

```

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

```

2127 \iow_now:Nn \@mainaux { \ExplSyntaxOn }
2128 \iow_now:Nn \@mainaux { \char_set_catcode_space:n { 32 } }
2129 \iow_now:Ne \@mainaux
2130 {
2131   \tl_gset:cn { c_@@_ \int_use:N \g_@@_env_int _ tl }
2132   { \exp_not:o \g_@@_aux_tl }
2133 }

```

```

2134 \iow_now:Nn \@mainaux { \ExplSyntaxOff }

2135 \bool_if:NT \g_@@_footnote_bool \endsavenotes
2136 }

```

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.

```

2137 \cs_new_protected:Npn \@_transform_preamble:
2138 {
2139   \@_transform_preamble_i:
2140   \@_transform_preamble_ii:
2141 }

2142 \cs_new_protected:Npn \@_transform_preamble_i:
2143 {
2144   \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).

```

2145 \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.
2146 \bool_gset_false:N \g_tmpb_bool

```

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

```

2147 \tl_gclear_new:N \g_@@_pre_cell_tl

The counter \l_tmpa_int will count the number of consecutive occurrences of the symbol |.
2148 \int_zero:N \l_tmpa_int
2149 \tl_gclear:N \g_@@_array_preamble_tl
2150 \tl_if_eq:NNTF \l_@@_vlines_clist \c_@@_all_tl
2151 {
2152   \tl_gset:Nn \g_@@_array_preamble_tl
2153   { ! { \skip_horizontal:N \arrayrulewidth } }
2154 }
2155 {
2156   \clist_if_in:NnT \l_@@_vlines_clist 1
2157   {
2158     \tl_gset:Nn \g_@@_array_preamble_tl
2159     { ! { \skip_horizontal:N \arrayrulewidth } }
2160   }
2161 }

```

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

```

2162 \exp_last_unbraced:No \@_rec_preamble:n \g_@@_user_preamble_tl \@_stop:
2163 \int_gset_eq:NN \g_@@_static_num_of_col_int \c@jCol

2164 \@_replace_columncolor:
2165 }

```

```

2166 \hook_gput_code:nnn { begindocument } { . }
2167 {
2168   \IfPackageLoadedTF { colortbl }
2169   {

```

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).

```

2170     \regex_const:Nn \c_@@_columncolor_regex { \c { columncolor } }
2171     \cs_new_protected:Npn \@@_replace_columncolor:
2172     {
2173       \regex_replace_all:NnN
2174       \c_@@_columncolor_regex
2175       { \c { @@_columncolor_preamble } }
2176       \g_@@_array_preamble_tl
2177     }
2178   }
2179   {
2180     \cs_new_protected:Npn \@@_replace_columncolor:
2181     { \cs_set_eq:NN \columncolor \@@_columncolor_preamble }
2182   }
2183 }

```

```

2184 \cs_new_protected:Npn \@@_transform_preamble_ii:
2185 {

```

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

```

2186   \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2187   {
2188     \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2189     { \bool_gset_true:N \g_@@_delims_bool }
2190   }
2191   { \bool_gset_true:N \g_@@_delims_bool }

```

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

```

2192   \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).

```

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

```

```

2216         {
2217             \bool_if:NF \l_@@_exterior_arraycolsep_bool
2218             { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { } } }
2219         }
2220     }
2221 }
2222 }

```

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`).

```

2223 \dim_compare:nNnT \l_@@_tabular_width_dim = \c_zero_dim
2224 {
2225     \tl_gput_right:Nn \g_@@_array_preamble_tl
2226     { > { \@@_error_too_much_cols: } 1 }
2227 }
2228 }

```

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.

```

2229 \cs_new_protected:Npn \@@_rec_preamble:n #1
2230 {

```

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

```

2231 \cs_if_exist:cTF { @@ _ \token_to_str:N #1 }
2232 { \use:c { @@ _ \token_to_str:N #1 } { #1 } }
2233 {

```

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

```

2234 \cs_if_exist:cTF { NC @ find @ #1 }
2235 {
2236     \tl_set_eq:Nc \l_tmpb_tl { NC @ rewrite @ #1 }
2237     \exp_last_unbraced:No \@@_rec_preamble:n \l_tmpb_tl
2238 }
2239 {

```

We use `\str_if_eq:nnTF` which is slightly faster than `\tl_if_eq:nnTF`.

```

2240 \str_if_eq:nnT { #1 } { S }
2241 { \@@_fatal:n { unknown-column-type-S } }
2242 { \@@_fatal:nn { unknown-column-type } { #1 } }
2243 }
2244 }
2245 }

```

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

```

2246 \cs_new:Npn \@@_c #1
2247 {
2248     \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2249     \tl_gclear:N \g_@@_pre_cell_tl
2250     \tl_gput_right:Nn \g_@@_array_preamble_tl
2251     { > \@@_cell_begin:w c < \@@_cell_end: }

```

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

```

2252 \int_gincr:N \c@jCol
2253 \@@_rec_preamble_after_col:n
2254 }

```

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`.

```

2255 \cs_new:Npn \@@_l #1
2256 {
2257   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2258   \tl_gclear:N \g_@@_pre_cell_tl
2259   \tl_gput_right:Nn \g_@@_array_preamble_tl
2260   {
2261     > { \@@_cell_begin:w \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_l_tl }
2262     l
2263     < \@@_cell_end:
2264   }
2265   \int_gincr:N \c@jCol
2266   \@@_rec_preamble_after_col:n
2267 }
2268 \cs_new:Npn \@@_r #1
2269 {
2270   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2271   \tl_gclear:N \g_@@_pre_cell_tl
2272   \tl_gput_right:Nn \g_@@_array_preamble_tl
2273   {
2274     > { \@@_cell_begin:w \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_r_tl }
2275     r
2276     < \@@_cell_end:
2277   }
2278   \int_gincr:N \c@jCol
2279   \@@_rec_preamble_after_col:n
2280 }

```

For ! and @

```

2281 \cs_new:cpn { @@ _ \token_to_str:N ! } #1 #2
2282 {
2283   \tl_gput_right:Nn \g_@@_array_preamble_tl { #1 { #2 } }
2284   \@@_rec_preamble:n
2285 }
2286 \cs_set_eq:cc { @@ _ \token_to_str:N @ } { @@ _ \token_to_str:N ! }

```

For |

```

2287 \cs_new:cpn { @@ _ | } #1
2288 {
2289   \l_tmpa_int is the number of successive occurrences of |
2290   \int_incr:N \l_tmpa_int
2291   \@@_make_preamble_i_i:n
2292 }
2292 \cs_new_protected:Npn \@@_make_preamble_i_i:n #1
2293 {
2294   \str_if_eq:nnTF { #1 } |
2295   { \use:c { @@ _ | } | }
2296   { \@@_make_preamble_i_ii:nn { } #1 }
2297 }
2298 \cs_new_protected:Npn \@@_make_preamble_i_ii:nn #1 #2
2299 {
2300   \str_if_eq:nnTF { #2 } [
2301   { \@@_make_preamble_i_iii:nn { #1 } [ }
2302   { \@@_make_preamble_i_iii:nn { #2 } { #1 } }
2303 }
2304 \cs_new_protected:Npn \@@_make_preamble_i_iii:nn #1 [ #2 ]
2305 { \@@_make_preamble_i_ii:nn { #1 , #2 } }
2306 \cs_new_protected:Npn \@@_make_preamble_i_iii:nn #1 #2
2307 {
2308   \@@_compute_rule_width:n { multiplicity = \l_tmpa_int , #2 }
2309   \tl_gput_right:Ne \g_@@_array_preamble_tl
2310   {

```

Here, the command `\dim_eval:n` is mandatory.

```

2311     \exp_not:N ! { \skip_horizontal:n { \dim_eval:n { \l_@@_rule_width_dim } } }
2312   }
2313   \tl_gput_right:Ne \g_@@_pre_code_after_tl
2314   {
2315     \@@_vline:n
2316     {
2317       position = \int_eval:n { \c@jCol + 1 } ,
2318       multiplicity = \int_use:N \l_tmpa_int ,
2319       total-width = \dim_use:N \l_@@_rule_width_dim ,
2320       #2
2321     }

```

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.

```

2322   }
2323   \int_zero:N \l_tmpa_int
2324   \str_if_eq:nnT { #1 } { \@@_stop: } { \bool_gset_true:N \g_tmpb_bool }
2325   \@@_rec_preamble:n #1
2326 }

2327 \cs_new:cpn { @@_ > } #1 #2
2328 {
2329   \tl_gput_right:Nn \g_@@_pre_cell_tl { > { #2 } }
2330   \@@_rec_preamble:n
2331 }

2332 \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.

```

2333 \keys_define:nn { nicematrix / p-column }
2334 {
2335   r .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_r_str ,
2336   r .value_forbidden:n = true ,
2337   c .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_c_str ,
2338   c .value_forbidden:n = true ,
2339   l .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_l_str ,
2340   l .value_forbidden:n = true ,
2341   R .code:n =
2342     \IfPackageLoadedTF { ragged2e }
2343     { \str_set_eq:NN \l_@@_hpos_col_str \c_@@_R_str }
2344     {
2345       \@@_error_or_warning:n { ragged2e~not~loaded }
2346       \str_set_eq:NN \l_@@_hpos_col_str \c_@@_r_str
2347     } ,
2348   R .value_forbidden:n = true ,
2349   L .code:n =
2350     \IfPackageLoadedTF { ragged2e }
2351     { \str_set_eq:NN \l_@@_hpos_col_str \c_@@_L_str }
2352     {
2353       \@@_error_or_warning:n { ragged2e~not~loaded }
2354       \str_set_eq:NN \l_@@_hpos_col_str \c_@@_l_str
2355     } ,
2356   L .value_forbidden:n = true ,
2357   C .code:n =
2358     \IfPackageLoadedTF { ragged2e }
2359     { \str_set_eq:NN \l_@@_hpos_col_str \c_@@_C_str }
2360     {
2361       \@@_error_or_warning:n { ragged2e~not~loaded }
2362       \str_set_eq:NN \l_@@_hpos_col_str \c_@@_c_str
2363     } ,

```

```

2364 C .value_forbidden:n = true ,
2365 S .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_si_str ,
2366 S .value_forbidden:n = true ,
2367 p .code:n = \str_set:Nn \l_@@_vpos_col_str { p } ,
2368 p .value_forbidden:n = true ,
2369 t .meta:n = p ,
2370 m .code:n = \str_set:Nn \l_@@_vpos_col_str { m } ,
2371 m .value_forbidden:n = true ,
2372 b .code:n = \str_set:Nn \l_@@_vpos_col_str { b } ,
2373 b .value_forbidden:n = true ,
2374 }

```

For p but also b and m.

```

2375 \cs_new:Npn \@@_p #1
2376 {
2377   \str_set:Nn \l_@@_vpos_col_str { #1 }

```

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

```

2378   \@@_make_preamble_ii_i:n
2379 }
2380 \cs_set_eq:NN \@@_b \@@_p
2381 \cs_set_eq:NN \@@_m \@@_p
2382 \cs_new_protected:Npn \@@_make_preamble_ii_i:n #1
2383 {
2384   \str_if_eq:nnTF { #1 } { [ ]
2385     { \@@_make_preamble_ii_ii:w [ ]
2386       { \@@_make_preamble_ii_ii:w [ ] { #1 } }
2387     }
2388   \cs_new_protected:Npn \@@_make_preamble_ii_ii:w [ #1 ]
2389     { \@@_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.

```

2390 \cs_new_protected:Npn \@@_make_preamble_ii_iii:nn #1 #2
2391 {

```

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).

```

2392   \str_set_eq:NN \l_@@_hpos_col_str \c_@@_j_str
2393   \@@_keys_p_column:n { #1 }
2394   \@@_make_preamble_ii_iv:nnn { #2 } { minipage } { }
2395 }
2396 \cs_new_protected:Npn \@@_keys_p_column:n #1
2397 { \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.

```

2398 \cs_new_protected:Npn \@@_make_preamble_ii_iv:nnn #1 #2 #3
2399 {
2400   \use:e
2401   {
2402     \@@_make_preamble_ii_v:nnnnnnnn
2403     { \str_if_eq:onTF \l_@@_vpos_col_str { p } { t } { b } }
2404     { \dim_eval:n { #1 } }
2405   }

```

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.

```

2406   \str_if_eq:NNTF \l_@@_hpos_col_str \c_@@_j_str
2407   { \tl_clear:N \exp_not:N \l_@@_hpos_cell_tl }

```



```

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

```

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

```

2433         { \str_lowercase:o \l_@@_hpos_col_str }
2434     }
2435 }

```

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

```

2436     \int_gincr:N \c@jCol
2437     \@@_rec_preamble_after_col:n
2438 }

```

#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*.

```

2439 \cs_new_protected:Npn \@@_make_preamble_ii_v:nnnnnnnn #1 #2 #3 #4 #5 #6 #7 #8
2440 {
2441     \tl_if_eq:NNTF \l_@@_hpos_col_str \c_@@_si_str
2442     {
2443         \tl_gput_right:Nn \g_@@_array_preamble_tl
2444         { > { \@@_test_if_empty_for_S: } }
2445     }
2446     {
2447         \tl_gput_right:Nn \g_@@_array_preamble_tl
2448         { > { \@@_test_if_empty: } }
2449     }
2450     \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2451     \tl_gclear:N \g_@@_pre_cell_tl
2452     \tl_gput_right:Nn \g_@@_array_preamble_tl

```

```

2453 {
2454 > {

```

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.

```

2455 \dim_set:Nn \l_@@_col_width_dim { #2 }
2456 \bool_if:NT \c_@@_testphase_table_bool
2457 { \tag_struct_begin:n { tag = Div } }
2458 \@@_cell_begin:w

```

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

```

2459 \use:c { #7 } [ #1 ] { #2 }

```

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

```

2460 \everypar
2461 {
2462 \vrule height \box_ht:N \@arstrutbox width \c_zero_dim
2463 \everypar { }
2464 }
2465 \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.).

```

2466 #3

```

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

```

2467 \g_@@_row_style_tl
2468 \arraybackslash
2469 #5
2470 }
2471 #8
2472 < {
2473 #6

```

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

```

2474 \@finalstrut \@arstrutbox
2475 \use:c { end #7 }

```

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

```

2476 #4
2477 \@@_cell_end:
2478 \bool_if:NT \c_@@_testphase_table_bool \tag_struct_end:
2479 }
2480 }
2481 }

```

```

2482 \str_new:N \c_@@_ignorespaces_str
2483 \str_set:Ne \c_@@_ignorespaces_str { \ignorespaces }
2484 \str_remove_all:Nn \c_@@_ignorespaces_str { ~ }
2485 \cs_new_protected:Npn \@@_test_if_empty:
2486 { \peek_after:Nw \@@_test_if_empty_i: }
2487 \cs_new_protected:Npn \@@_test_if_empty_i:
2488 {
2489 \str_set:Ne \l_tmpa_str { \token_to_meaning:N \l_peek_token }
2490 \str_if_eq:NNT \l_tmpa_str \c_@@_ignorespaces_str
2491 { \@@_test_if_empty:w }
2492 }
2493 \cs_new_protected:Npn \@@_test_if_empty:w \ignorespaces
2494 { \peek_after:Nw \@@_test_if_empty_ii: }

```

```

2495 \cs_new_protected:Npn \@@_nullify_cell:
2496 {
2497 \tl_gput_right:Nn \g_@@_cell_after_hook_tl

```

```

2498     {
2499       \box_set_wd:Nn \l_@@_cell_box \c_zero_dim
2500       \skip_horizontal:N \l_@@_col_width_dim
2501     }
2502   }

2503 \bool_if:NTF \c_@@_tagging_array_bool
2504 {
2505   \cs_new_protected:Npn \@@_test_if_empty_ii:
2506     { \peek_meaning:NT \textonly@unskip \@@_nullify_cell: }
2507 }

```

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

```

2508 {
2509   \cs_new_protected:Npn \@@_test_if_empty_ii:
2510     { \peek_meaning:NT \unskip \@@_nullify_cell: }
2511 }

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

```

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.

```

2520 \cs_new_protected:Npn \@@_center_cell_box:
2521 {

```

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

```

2522   \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2523   {
2524     \int_compare:nNnT
2525       { \box_ht:N \l_@@_cell_box }
2526     >

```

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).

```

2527     { \box_ht:N \strutbox }
2528     {
2529       \hbox_set:Nn \l_@@_cell_box
2530       {
2531         \box_move_down:nn
2532         {
2533           ( \box_ht:N \l_@@_cell_box - \box_ht:N \@arstrutbox
2534             + \baselineskip ) / 2
2535         }
2536         { \box_use:N \l_@@_cell_box }
2537       }
2538     }
2539   }
2540 }

```

For V (similar to the V of varwidth).

```

2541 \cs_new:Npn \@@_V #1 #2
2542 {
2543   \str_if_eq:nnTF { #2 } { [ ] }
2544     { \@@_make_preamble_V_i:w [ ] }
2545     { \@@_make_preamble_V_i:w [ ] { #2 } }
2546 }
2547 \cs_new_protected:Npn \@@_make_preamble_V_i:w [ #1 ]
2548 { \@@_make_preamble_V_ii:nn { #1 } }
2549 \cs_new_protected:Npn \@@_make_preamble_V_ii:nn #1 #2
2550 {
2551   \str_set:Nn \l_@@_vpos_col_str { p }
2552   \str_set_eq:NN \l_@@_hpos_col_str \c_@@_j_str
2553   \@@_keys_p_column:n { #1 }
2554   \IfPackageLoadedTF { varwidth }
2555     { \@@_make_preamble_ii_iv:nnn { #2 } { varwidth } { } }
2556     {
2557       \@@_error_or_warning:n { varwidth-not-loaded }
2558       \@@_make_preamble_ii_iv:nnn { #2 } { minipage } { }
2559     }
2560 }

```

For w and W

```

2561 \cs_new:Npn \@@_w { \@@_make_preamble_w:nnnn { } }
2562 \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.

```

2563 \cs_new_protected:Npn \@@_make_preamble_w:nnnn #1 #2 #3 #4
2564 {
2565   \str_if_eq:nnTF { #3 } { s }
2566     { \@@_make_preamble_w_i:nnnn { #1 } { #4 } }
2567     { \@@_make_preamble_w_ii:nnnn { #1 } { #2 } { #3 } { #4 } }
2568 }

```

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.

```

2569 \cs_new_protected:Npn \@@_make_preamble_w_i:nnnn #1 #2
2570 {
2571   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2572   \tl_gclear:N \g_@@_pre_cell_tl
2573   \tl_gput_right:Nn \g_@@_array_preamble_tl
2574     {
2575       > {
2576         \dim_set:Nn \l_@@_col_width_dim { #2 }
2577         \@@_cell_begin:w
2578         \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_c_tl
2579       }
2580       c
2581       < {
2582         \@@_cell_end_for_w_s:
2583         #1
2584         \@@_adjust_size_box:
2585         \box_use_drop:N \l_@@_cell_box
2586       }
2587     }
2588   \int_gincr:N \c@jCol
2589   \@@_rec_preamble_after_col:n
2590 }

```

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

```

2591 \cs_new_protected:Npn \@@_make_preamble_w_ii:nnnn #1 #2 #3 #4
2592 {
2593   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2594   \tl_gclear:N \g_@@_pre_cell_tl
2595   \tl_gput_right:Nn \g_@@_array_preamble_tl
2596   {
2597     > {

```

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.

```

2598       \dim_set:Nn \l_@@_col_width_dim { #4 }
2599       \hbox_set:Nw \l_@@_cell_box
2600       \@@_cell_begin:w
2601       \cs_set_nopar:Npn \l_@@_hpos_cell_tl { #3 }
2602     }
2603     c
2604     < {
2605       \@@_cell_end:
2606       \hbox_set_end:
2607       #1
2608       \@@_adjust_size_box:
2609       \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2610     }
2611   }

```

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

```

2612   \int_gincr:N \c@jCol
2613   \@@_rec_preamble_after_col:n
2614 }

```

```

2615 \cs_new_protected:Npn \@@_special_W:
2616 {
2617   \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \l_@@_col_width_dim
2618   { \@@_warning:n { W~warning } }
2619 }

```

For S (of siunitx).

```

2620 \cs_new:Npn \@@_S #1 #2
2621 {
2622   \str_if_eq:nnTF { #2 } { [ ] }
2623   { \@@_make_preamble_S:w [ ] }
2624   { \@@_make_preamble_S:w [ ] { #2 } }
2625 }
2626 \cs_new_protected:Npn \@@_make_preamble_S:w [ #1 ]
2627 { \@@_make_preamble_S:i:n { #1 } }
2628 \cs_new_protected:Npn \@@_make_preamble_S:i:n #1
2629 {
2630   \IfPackageLoadedF { siunitx } { \@@_fatal:n { siunitx~not~loaded } }
2631   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2632   \tl_gclear:N \g_@@_pre_cell_tl
2633   \tl_gput_right:Nn \g_@@_array_preamble_tl
2634   {
2635     > {
2636       \@@_cell_begin:w
2637       \keys_set:nn { siunitx } { #1 }
2638       \siunitx_cell_begin:w
2639     }
2640     c
2641     < { \siunitx_cell_end: \@@_cell_end: }
2642   }

```

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

```

2643 \int_gincr:N \c@jCol
2644 \@@_rec_preamble_after_col:n
2645 }

```

For (, [and \{.

```

2646 \cs_new:cpn { @@ _ \token_to_str:N ( } #1 #2
2647 {
2648 \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.

```

2649 \int_if_zero:nTF \c@jCol
2650 {
2651 \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2652 {

```

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

```

2653 \tl_gset:Nn \g_@@_left_delim_tl { #1 }
2654 \tl_gset_eq:NN \g_@@_right_delim_tl \c_@@_dot_tl
2655 \@@_rec_preamble:n #2
2656 }
2657 {
2658 \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2659 \@@_make_preamble_iv:nn { #1 } { #2 }
2660 }
2661 }
2662 { \@@_make_preamble_iv:nn { #1 } { #2 } }
2663 }
2664 \cs_set_eq:cc { @@ _ \token_to_str:N [ ] { @@ _ \token_to_str:N ( }
2665 \cs_set_eq:cc { @@ _ \token_to_str:N \{ } { @@ _ \token_to_str:N ( }
2666 \cs_new_protected:Npn \@@_make_preamble_iv:nn #1 #2
2667 {
2668 \tl_gput_right:Ne \g_@@_pre_code_after_tl
2669 { \@@_delimiter:nnn #1 { \int_eval:n { \c@jCol + 1 } } \c_true_bool }
2670 \tl_if_in:nnTF { ( [ \{ ) ] \} \left \right } { #2 }
2671 {
2672 \@@_error:nn { delimiter~after~opening } { #2 }
2673 \@@_rec_preamble:n
2674 }
2675 { \@@_rec_preamble:n #2 }
2676 }

```

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

```

2677 \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}).

```

2678 \cs_new:cpn { @@ _ \token_to_str:N ) } #1 #2
2679 {
2680 \bool_if:NT \l_@@_small_bool { \@@_fatal:n { Delimiter~with~small } }
2681 \tl_if_in:nnTF { ) ] \} } { #2 }
2682 { \@@_make_preamble_v:nnn #1 #2 }
2683 {
2684 \str_if_eq:nnTF { \@@_stop: } { #2 }
2685 {
2686 \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2687 { \tl_gset:Nn \g_@@_right_delim_tl { #1 } }
2688 {
2689 \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2690 \tl_gput_right:Ne \g_@@_pre_code_after_tl

```

```

2691         { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2692         \@@_rec_preamble:n #2
2693     }
2694 }
2695 {
2696     \tl_if_in:nnT { ( [ \{ \left } { #2 }
2697         { \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } } }
2698     \tl_gput_right:Ne \g_@@_pre_code_after_tl
2699         { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2700     \@@_rec_preamble:n #2
2701 }
2702 }
2703 }
2704 \cs_set_eq:cc { @@ _ \token_to_str:N ] } { @@ _ \token_to_str:N ) }
2705 \cs_set_eq:cc { @@ _ \token_to_str:N \} } { @@ _ \token_to_str:N ) }
2706 \cs_new_protected:Npn \@@_make_preamble_v:nnn #1 #2 #3
2707 {
2708     \str_if_eq:nnTF { \@@_stop: } { #3 }
2709     {
2710         \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2711         {
2712             \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2713             \tl_gput_right:Ne \g_@@_pre_code_after_tl
2714                 { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2715             \tl_gset:Nn \g_@@_right_delim_tl { #2 }
2716         }
2717         {
2718             \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2719             \tl_gput_right:Ne \g_@@_pre_code_after_tl
2720                 { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2721             \@@_error:nn { double~closing~delimiter } { #2 }
2722         }
2723     }
2724     {
2725         \tl_gput_right:Ne \g_@@_pre_code_after_tl
2726             { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2727         \@@_error:nn { double~closing~delimiter } { #2 }
2728         \@@_rec_preamble:n #3
2729     }
2730 }
2731 \cs_new:cpn { @@ _ \token_to_str:N \right } #1
2732 { \use:c { @@ _ \token_to_str:N ) } }

```

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

```

2733 \cs_new_protected:Npn \@@_rec_preamble_after_col:n #1
2734 {
2735     \str_if_eq:nnTF { #1 } { < }
2736     \@@_rec_preamble_after_col_i:n
2737     {
2738         \str_if_eq:nnTF { #1 } { @ }
2739         \@@_rec_preamble_after_col_ii:n
2740         {
2741             \tl_if_eq:NNTF \l_@@_vlines_clist \c_@@_all_tl
2742             {
2743                 \tl_gput_right:Nn \g_@@_array_preamble_tl
2744                     { ! { \skip_horizontal:N \arrayrulewidth } }
2745             }
2746             {
2747                 \clist_if_in:NeT \l_@@_vlines_clist { \int_eval:n { \c@jCol + 1 } } }

```

```

2748         {
2749             \tl_gput_right:Nn \g_@@_array_preamble_tl
2750             { ! { \skip_horizontal:N \arrayrulewidth } }
2751         }
2752     }
2753     \@@_rec_preamble:n { #1 }
2754 }
2755 }
2756 }
2757 \cs_new_protected:Npn \@@_rec_preamble_after_col_i:n #1
2758 {
2759     \tl_gput_right:Nn \g_@@_array_preamble_tl { < { #1 } }
2760     \@@_rec_preamble_after_col:n
2761 }

```

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

```

2762 \cs_new_protected:Npn \@@_rec_preamble_after_col_ii:n #1
2763 {
2764     \tl_if_eq:NNTF \l_@@_vlines_clist \c_@@_all_tl
2765     {
2766         \tl_gput_right:Nn \g_@@_array_preamble_tl
2767         { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2768     }
2769     {
2770         \clist_if_in:NcTF \l_@@_vlines_clist { \int_eval:n { \c@jCol + 1 } }
2771         {
2772             \tl_gput_right:Nn \g_@@_array_preamble_tl
2773             { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2774         }
2775         { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { #1 } } }
2776     }
2777     \@@_rec_preamble:n
2778 }
2779 \cs_new:cpn { @@ _ * } #1 #2 #3
2780 {
2781     \tl_clear:N \l_tmpa_tl
2782     \int_step_inline:nn { #2 } { \tl_put_right:Nn \l_tmpa_tl { #3 } }
2783     \exp_last_unbraced:No \@@_rec_preamble:n \l_tmpa_tl
2784 }

```

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.

```

2785 \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`.

```

2786 \cs_new:Npn \@@_X #1 #2
2787 {
2788     \str_if_eq:nnTF { #2 } { [ ]
2789     { \@@_make_preamble_X:w [ ] }
2790     { \@@_make_preamble_X:w [ ] #2 }
2791 }
2792 \cs_new_protected:Npn \@@_make_preamble_X:w [ #1 ]
2793 { \@@_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`).


```

2794 \keys_define:nn { nicematrix / X-column }
2795 { 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.

```

2796 \cs_new_protected:Npn \@@_make_preamble_X_i:n #1
2797 {

```

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).

```

2798 \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).

```

2799 \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`.

```

2800 \int_zero_new:N \l_@@_weight_int
2801 \int_set_eq:NN \l_@@_weight_int \c_one_int
2802 \@@_keys_p_column:n { #1 }

```

The unknown keys are put in `\l_tmpa_tl`

```

2803 \keys_set:no { nicematrix / X-column } \l_tmpa_tl
2804 \int_compare:nNnT \l_@@_weight_int < \c_zero_int
2805 {
2806   \@@_error_or_warning:n { negative-weight }
2807   \int_set:Nn \l_@@_weight_int { - \l_@@_weight_int }
2808 }
2809 \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).

```

2810 \bool_if:NTF \l_@@_X_columns_aux_bool
2811 {
2812   \@@_make_preamble_ii_iv:nnn
2813   { \l_@@_weight_int \l_@@_X_columns_dim }
2814   { minipage }
2815   { \@@_no_update_width: }
2816 }
2817 {
2818   \tl_gput_right:Nn \g_@@_array_preamble_tl
2819   {
2820     > {
2821       \@@_cell_begin:w
2822       \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`.

```

2823 \NotEmpty

```

The following code will nullify the box of the cell.

```

2824 \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2825 { \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`.

```

2826 \begin { minipage } { 5 cm } \arraybackslash
2827 }
2828 c
2829 < {
2830 \end { minipage }
2831 \@@_cell_end:

```

```

2832     }
2833   }
2834   \int_gincr:N \c@jCol
2835   \@@_rec_preamble_after_col:n
2836 }
2837 }

2838 \cs_new_protected:Npn \@@_no_update_width:
2839 {
2840   \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2841   { \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing: }
2842 }

```

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

```

2843 \cs_new_protected:Npn \@@_make_preamble_vlism:n #1
2844 {
2845   \seq_gput_right:Ne \g_@@_cols_vlism_seq
2846   { \int_eval:n { \c@jCol + 1 } }
2847   \tl_gput_right:Ne \g_@@_array_preamble_tl
2848   { \exp_not:N ! { \skip_horizontal:N \arrayrulewidth } }
2849   \@@_rec_preamble:n
2850 }

```

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.

```

2851 \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).

```

2852 \cs_new_protected:cpn { @@ _ \token_to_str:N \hline }
2853 { \@@_fatal:n { Preamble-forgotten } }
2854 \cs_set_eq:cc { @@ _ \token_to_str:N \Hline } { @@ _ \token_to_str:N \hline }
2855 \cs_set_eq:cc { @@ _ \token_to_str:N \toprule } { @@ _ \token_to_str:N \hline }
2856 \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).

```

2857 \cs_new:Npn \@@_multicolumn:nnn #1 #2 #3
2858 {

```

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`.

```

2859   \multispan { #1 }
2860   \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing:
2861   \begingroup
2862   \bool_if:NT \c_@@_testphase_table_bool
2863   { \tbl_update_multicolumn_cell_data:n { #1 } }
2864   \cs_set_nopar:Npn \@addamp
2865   { \legacy_if:nTF { @firstamp } { \@firstampfalse } { \@preamerr 5 } }

```

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

```

2866   \tl_gclear:N \g_@@_preamble_tl
2867   \@@_make_m_preamble:n #2 \q_stop

```

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

```
2868 \exp_args:No \mkpream \g_@@_preamble_tl
2869 \@addtopreamble \empty
2870 \endgroup
2871 \bool_if:NT \c_@@_testphase_table_bool
2872 { \UseTaggingSocket { tbl / colspan } { #1 } }
```

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

```
2873 \int_compare:nNtT { #1 } > \c_one_int
2874 {
2875   \seq_gput_left:Ne \g_@@_multicolumn_cells_seq
2876   { \int_use:N \c@iRow - \int_eval:n { \c@jCol + 1 } }
2877   \seq_gput_left:Nn \g_@@_multicolumn_sizes_seq { #1 }
2878   \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
2879   {
2880     {
2881       \int_if_zero:nTF \c@jCol
2882       { \int_eval:n { \c@iRow + 1 } }
2883       { \int_use:N \c@iRow }
2884     }
2885     { \int_eval:n { \c@jCol + 1 } }
2886     {
2887       \int_if_zero:nTF \c@jCol
2888       { \int_eval:n { \c@iRow + 1 } }
2889       { \int_use:N \c@iRow }
2890     }
2891     { \int_eval:n { \c@jCol + #1 } }
2892     { } % for the name of the block
2893   }
2894 }
```

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

```
2895 \RenewDocumentCommand \cellcolor { 0 { } m }
2896 {
2897   \@@_test_color_inside:
2898   \tl_gput_right:Ne \g_@@_pre_code_before_tl
2899   {
2900     \@@_rectanglecolor [ ##1 ]
2901     { \exp_not:n { ##2 } }
2902     { \int_use:N \c@iRow - \int_use:N \c@jCol }
2903     { \int_use:N \c@iRow - \int_eval:n { \c@jCol + #1 } }
2904   }
2905   \ignorespaces
2906 }
```

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

```
2907 \cs_set_nopar:Npn \@sharp { #3 }
2908 \@arstrut
2909 \@preamble
2910 \null
```

We add some lines.

```
2911 \int_gadd:Nn \c@jCol { #1 - 1 }
2912 \int_compare:nNtT \c@jCol > \g_@@_col_total_int
2913 { \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
2914 \ignorespaces
2915 }
```

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`.

```

2916 \cs_new_protected:Npn \@@_make_m_preamble:n #1
2917 {
2918   \str_case:nnF { #1 }
2919   {
2920     c { \@@_make_m_preamble_i:n #1 }
2921     l { \@@_make_m_preamble_i:n #1 }
2922     r { \@@_make_m_preamble_i:n #1 }
2923     > { \@@_make_m_preamble_ii:nn #1 }
2924     ! { \@@_make_m_preamble_ii:nn #1 }
2925     @ { \@@_make_m_preamble_ii:nn #1 }
2926     | { \@@_make_m_preamble_iii:n #1 }
2927     p { \@@_make_m_preamble_iv:nnn t #1 }
2928     m { \@@_make_m_preamble_iv:nnn c #1 }
2929     b { \@@_make_m_preamble_iv:nnn b #1 }
2930     w { \@@_make_m_preamble_v:nnnn { } #1 }
2931     W { \@@_make_m_preamble_v:nnnn { \@@_special_W: } #1 }
2932     \q_stop { }
2933   }
2934   {
2935     \cs_if_exist:cTF { NC @ find @ #1 }
2936     {
2937       \tl_set_eq:Nc \l_tmpa_tl { NC @ rewrite @ #1 }
2938       \exp_last_unbraced:No \@@_make_m_preamble:n \l_tmpa_tl
2939     }
2940     {

```

We use `\str_if_eq:nnTF` which is slightly faster than `\tl_if_eq:nnTF`.

```

2941       \str_if_eq:nnT { #1 } { S }
2942       { \@@_fatal:n { unknown~column~type~S } }
2943       { \@@_fatal:nn { unknown~column~type } { #1 } }
2944     }
2945   }
2946 }

```

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

```

2947 \cs_new_protected:Npn \@@_make_m_preamble_i:n #1
2948 {
2949   \tl_gput_right:Nn \g_@@_preamble_tl
2950   {
2951     > { \@@_cell_begin:w \cs_set_nopar:Npn \l_@@_hpos_cell_tl { #1 } }
2952     #1
2953     < \@@_cell_end:
2954   }

```

We test for the presence of a `<`.

```

2955   \@@_make_m_preamble_x:n
2956 }

```

For `>`, `!` and `@`

```

2957 \cs_new_protected:Npn \@@_make_m_preamble_ii:nn #1 #2
2958 {
2959   \tl_gput_right:Nn \g_@@_preamble_tl { #1 { #2 } }
2960   \@@_make_m_preamble:n
2961 }

```

For `|`

```

2962 \cs_new_protected:Npn \@@_make_m_preamble_iii:n #1
2963 {
2964   \tl_gput_right:Nn \g_@@_preamble_tl { #1 }
2965   \@@_make_m_preamble:n
2966 }

```

For p, m and b

```

2967 \cs_new_protected:Npn \@@_make_m_preamble_iv:nnn #1 #2 #3
2968 {
2969   \tl_gput_right:Nn \g_@@_preamble_tl
2970   {
2971     > {
2972       \@@_cell_begin:w
2973       \begin { minipage } [ #1 ] { \dim_eval:n { #3 } }
2974       \mode_leave_vertical:
2975       \arraybackslash
2976       \vrule height \box_ht:N \@@arstrutbox depth 0 pt width 0 pt
2977     }
2978     c
2979     < {
2980       \vrule height 0 pt depth \box_dp:N \@@arstrutbox width 0 pt
2981       \end { minipage }
2982       \@@_cell_end:
2983     }
2984   }

```

We test for the presence of a <.

```

2985   \@@_make_m_preamble_x:n
2986 }

```

For w and W

```

2987 \cs_new_protected:Npn \@@_make_m_preamble_v:nnnn #1 #2 #3 #4
2988 {
2989   \tl_gput_right:Nn \g_@@_preamble_tl
2990   {
2991     > {
2992       \dim_set:Nn \l_@@_col_width_dim { #4 }
2993       \hbox_set:Nw \l_@@_cell_box
2994       \@@_cell_begin:w
2995       \cs_set_nopar:Npn \l_@@_hpos_cell_tl { #3 }
2996     }
2997     c
2998     < {
2999       \@@_cell_end:
3000       \hbox_set_end:
3001       \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
3002       #1
3003       \@@_adjust_size_box:
3004       \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
3005     }
3006   }

```

We test for the presence of a <.

```

3007   \@@_make_m_preamble_x:n
3008 }

```

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

```

3009 \cs_new_protected:Npn \@@_make_m_preamble_x:n #1
3010 {
3011   \str_if_eq:nnTF { #1 } { < }
3012   \@@_make_m_preamble_ix:n
3013   { \@@_make_m_preamble:n { #1 } }
3014 }
3015 \cs_new_protected:Npn \@@_make_m_preamble_ix:n #1
3016 {
3017   \tl_gput_right:Nn \g_@@_preamble_tl { < { #1 } }
3018   \@@_make_m_preamble_x:n
3019 }

```

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`).

```

3020 \cs_new_protected:Npn \@@_put_box_in_flow:
3021 {
3022   \box_set_ht:Nn \l_tmpa_box { \box_ht:N \l_tmpa_box + \l_tmpa_dim }
3023   \box_set_dp:Nn \l_tmpa_box { \box_dp:N \l_tmpa_box + \l_tmpb_dim }
3024   \tl_if_eq:NNTF \l_@@_baseline_tl \c_@@_c_tl
3025     { \box_use_drop:N \l_tmpa_box }
3026   \@@_put_box_in_flow_i:
3027 }

```

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).

```

3028 \cs_new_protected:Npn \@@_put_box_in_flow_i:
3029 {
3030   \pgfpicture
3031     \@@_qpoint:n { row - 1 }
3032     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3033     \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
3034     \dim_gadd:Nn \g_tmpa_dim \pgf@y
3035     \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).

```

3036   \tl_if_in:NnTF \l_@@_baseline_tl { line- }
3037   {
3038     \int_set:Nn \l_tmpa_int
3039     {
3040       \str_range:Nnn
3041         \l_@@_baseline_tl
3042         6
3043         { \tl_count:o \l_@@_baseline_tl }
3044     }
3045     \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
3046   }
3047   {
3048     \tl_if_eq:NnTF \l_@@_baseline_tl { t }
3049     { \int_set_eq:NN \l_tmpa_int \c_one_int }
3050     {
3051       \tl_if_eq:NnTF \l_@@_baseline_tl { b }
3052       { \int_set_eq:NN \l_tmpa_int \c@iRow }
3053       { \int_set:Nn \l_tmpa_int \l_@@_baseline_tl }
3054     }
3055     \bool_lazy_or:nnT
3056     { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
3057     { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
3058     {
3059       \@@_error:n { bad-value-for-baseline }
3060       \int_set_eq:NN \l_tmpa_int \c_one_int
3061     }
3062     \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }

```

We take into account the position of the mathematical axis.

```

3063     \dim_gsub:Nn \g_tmpa_dim { \fontdimen22 \textfont2 }
3064   }
3065   \dim_gsub:Nn \g_tmpa_dim \pgf@y

```

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

```

3066   \endpgfpicture
3067   \box_move_up:nn \g_tmpa_dim { \box_use_drop:N \l_tmpa_box }
3068   \box_use_drop:N \l_tmpa_box
3069 }

```

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).

```
3070 \cs_new_protected:Npn \@@_use_arraybox_with_notes_c:
3071 {
```

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.

```
3072 \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3073 {
3074   \int_compare:nNt \c@jCol > \c_one_int
3075   {
3076     \box_set_wd:Nn \l_@@_the_array_box
3077     { \box_wd:N \l_@@_the_array_box - \arraycolsep }
3078   }
3079 }
```

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

```
3080 \begin { minipage } [ t ] { \box_wd:N \l_@@_the_array_box }
3081 \bool_if:NT \l_@@_caption_above_bool
3082 {
3083   \tl_if_empty:NF \l_@@_caption_tl
3084   {
3085     \bool_set_false:N \g_@@_caption_finished_bool
3086     \int_gzero:N \c@tabularnote
3087     \@@_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).

```
3088   \int_compare:nNt \g_@@_notes_caption_int > \c_zero_int
3089   {
3090     \tl_gput_right:Ne \g_@@_aux_tl
3091     {
3092       \tl_set:Nn \exp_not:N \l_@@_note_in_caption_tl
3093       { \int_use:N \g_@@_notes_caption_int }
3094     }
3095     \int_gzero:N \g_@@_notes_caption_int
3096   }
3097 }
3098 }
```

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

```
3099 \hbox
3100 {
3101   \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.

```
3102 \@@_create_extra_nodes:
3103 \seq_if_empty:NF \g_@@_blocks_seq \@@_draw_blocks:
3104 }
```

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

```
3105 \bool_lazy_any:nT
3106 {
3107   { ! \seq_if_empty_p:N \g_@@_notes_seq }
```

```

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

```

```

3116 \cs_new_protected:Npn \@@_insert_caption:
3117 {
3118   \tl_if_empty:NF \l_@@_caption_tl
3119   {
3120     \cs_if_exist:NTF \@cptype
3121     { \@@_insert_caption_i: }
3122     { \@@_error:n { caption~outside~float } }
3123   }
3124 }

```

```

3125 \cs_new_protected:Npn \@@_insert_caption_i:
3126 {
3127   \group_begin:

```

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

```

3128   \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.

```

3129   \IfPackageLoadedT { floatrow }
3130   { \cs_set_eq:NN \@makecaption \FR@makecaption }
3131   \tl_if_empty:NTF \l_@@_short_caption_tl
3132   { \caption }
3133   { \caption [ \l_@@_short_caption_tl ] }
3134   { \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.

```

3135   \bool_if:NF \g_@@_caption_finished_bool
3136   {
3137     \bool_gset_true:N \g_@@_caption_finished_bool
3138     \int_gset_eq:NN \g_@@_notes_caption_int \c@tabularnote
3139     \int_gzero:N \c@tabularnote
3140   }
3141   \tl_if_empty:NF \l_@@_label_tl { \label { \l_@@_label_tl } }
3142   \group_end:
3143 }

```

```

3144 \cs_new_protected:Npn \@@_tabularnote_error:n #1
3145 {
3146   \@@_error_or_warning:n { tabularnote~below~the~tabular }
3147   \@@_gredirect_none:n { tabularnote~below~the~tabular }
3148 }

```

```

3149 \cs_new_protected:Npn \@@_insert_tabularnotes:
3150 {
3151   \seq_gconcat:NNN \g_@@_notes_seq \g_@@_notes_in_caption_seq \g_@@_notes_seq
3152   \int_set:Nn \c@tabularnote { \seq_count:N \g_@@_notes_seq }
3153   \skip_vertical:N 0.65ex

```


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

```

3154 \group_begin:
3155 \l_@@_notes_code_before_tl
3156 \tl_if_empty:NF \g_@@_tabularnote_tl
3157 {
3158   \g_@@_tabularnote_tl \par
3159   \tl_gclear:N \g_@@_tabularnote_tl
3160 }

```

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.

```

3161 \int_compare:nNnT \c@tabularnote > \c_zero_int
3162 {
3163   \bool_if:NTF \l_@@_notes_para_bool
3164   {
3165     \begin { tabularnotes* }
3166     \seq_map_inline:Nn \g_@@_notes_seq
3167       { \@@_one_tabularnote:nn ##1 }
3168     \strut
3169     \end { tabularnotes* }

```

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

```

3170 \par
3171 }
3172 {
3173   \tabularnotes
3174   \seq_map_inline:Nn \g_@@_notes_seq
3175     { \@@_one_tabularnote:nn ##1 }
3176   \strut
3177   \endtabularnotes
3178 }
3179 }
3180 \unskip
3181 \group_end:
3182 \bool_if:NT \l_@@_notes_bottomrule_bool
3183 {
3184   \IfPackageLoadedTF { booktabs }
3185   {

```

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

```

3186 \skip_vertical:N \aboverulesep
\CT@arc@ is the specification of color defined by colortbl but you use it even if colortbl is not loaded.
3187 { \CT@arc@ \hrule height \heavyrulewidth }
3188 }
3189 { \@@_error_or_warning:n { bottomrule-without-booktabs } }
3190 }
3191 \l_@@_notes_code_after_tl
3192 \seq_gclear:N \g_@@_notes_seq
3193 \seq_gclear:N \g_@@_notes_in_caption_seq
3194 \int_gzero:N \c@tabularnote
3195 }

```

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.

```

3196 \cs_set_protected:Npn \@@_one_tabularnote:nn #1
3197 {
3198   \tl_if_novalue:nTF { #1 }
3199     { \item }
3200     { \item [ \@@_notes_label_in_list:n { #1 } ] }
3201 }

```

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`.

```

3202 \cs_new_protected:Npn \@@_use_arraybox_with_notes_b:
3203 {
3204   \pgfpicture
3205     \@@_qpoint:n { row - 1 }
3206     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3207     \@@_qpoint:n { row - \int_use:N \c@iRow - base }
3208     \dim_gsub:Nn \g_tmpa_dim \pgf@y
3209   \endpgfpicture
3210   \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3211   \int_if_zero:nT \l_@@_first_row_int
3212   {
3213     \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3214     \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3215   }
3216   \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3217 }

```

Now, the general case.

```

3218 \cs_new_protected:Npn \@@_use_arraybox_with_notes:
3219 {

```

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

```

3220   \tl_if_eq:NnT \l_@@_baseline_tl { t }
3221   { \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`.

```

3222   \pgfpicture
3223   \@@_qpoint:n { row - 1 }
3224   \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3225   \str_if_in:NnTF \l_@@_baseline_tl { line- }
3226   {
3227     \int_set:Nn \l_tmpa_int
3228     {
3229       \str_range:Nnn
3230         \l_@@_baseline_tl
3231         6
3232         { \tl_count:o \l_@@_baseline_tl }
3233     }
3234     \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
3235   }
3236   {
3237     \int_set:Nn \l_tmpa_int \l_@@_baseline_tl
3238     \bool_lazy_or:nnT
3239     { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
3240     { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
3241     {
3242       \@@_error:n { bad~value~for~baseline }
3243       \int_set:Nn \l_tmpa_int 1
3244     }
3245     \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }
3246   }
3247   \dim_gsub:Nn \g_tmpa_dim \pgf@y
3248   \endpgfpicture
3249   \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3250   \int_if_zero:nT \l_@@_first_row_int
3251   {
3252     \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3253     \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3254   }
3255   \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }

```

3256 }

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.

```
3257 \cs_new_protected:Npn \@@_put_box_in_flow_bis:nn #1 #2
3258 {
```

We will compute the real width of both delimiters used.

```
3259 \dim_zero_new:N \l_@@_real_left_delim_dim
3260 \dim_zero_new:N \l_@@_real_right_delim_dim
3261 \hbox_set:Nn \l_tmpb_box
3262 {
3263   \c_math_toggle_token
3264   \left #1
3265   \vcenter
3266   {
3267     \vbox_to_ht:nn
3268     { \box_ht_plus_dp:N \l_tmpa_box }
3269     { }
3270   }
3271   \right .
3272   \c_math_toggle_token
3273 }
3274 \dim_set:Nn \l_@@_real_left_delim_dim
3275 { \box_wd:N \l_tmpb_box - \nulldelimiterspace }
3276 \hbox_set:Nn \l_tmpb_box
3277 {
3278   \c_math_toggle_token
3279   \left .
3280   \vbox_to_ht:nn
3281   { \box_ht_plus_dp:N \l_tmpa_box }
3282   { }
3283   \right #2
3284   \c_math_toggle_token
3285 }
3286 \dim_set:Nn \l_@@_real_right_delim_dim
3287 { \box_wd:N \l_tmpb_box - \nulldelimiterspace }
```

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

```
3288 \skip_horizontal:N \l_@@_left_delim_dim
3289 \skip_horizontal:N -\l_@@_real_left_delim_dim
3290 \@@_put_box_in_flow:
3291 \skip_horizontal:N \l_@@_right_delim_dim
3292 \skip_horizontal:N -\l_@@_real_right_delim_dim
3293 }
```

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).

```
3294 \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).

```
3295 {
3296   \peek_remove_spaces:n
3297   {
3298     \peek_meaning:NTF \end
3299     \@@_analyze_end:Nn
```

```

3300     {
3301         \@@_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).
3302         \@@_array:o \g_@@_array_preamble_tl
3303     }
3304 }
3305 }
3306 {
3307     \@@_create_col_nodes:
3308     \endarray
3309 }

```

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

```

3310 \NewDocumentEnvironment { @@-light-syntax } { b }
3311 {

```

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`.

```

3312     \tl_if_empty:nT { #1 } { \@@_fatal:n { empty~environment } }
3313     \tl_map_inline:nn { #1 }
3314     {
3315         \str_if_eq:nnT { ##1 } { & }
3316         { \@@_fatal:n { ampersand-in~light-syntax } }
3317         \str_if_eq:nnT { ##1 } { \ }
3318         { \@@_fatal:n { double-backslash-in~light-syntax } }
3319     }

```

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`.

```

3320     \@@_light_syntax_i:w #1 \CodeAfter \q_stop

```

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

```

3321 }

```

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.

```

3322 {
3323     \@@_create_col_nodes:
3324     \endarray
3325 }
3326 \cs_new_protected:Npn \@@_light_syntax_i:w #1\CodeAfter #2\q_stop
3327 {
3328     \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).

```

3329     \seq_clear_new:N \l_@@_rows_seq

```

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

```

3330     \tl_set_rescan:Nno \l_@@_end_of_row_tl { } \l_@@_end_of_row_tl
3331     \bool_if:NTF \l_@@_light_syntax_expanded_bool
3332         \seq_set_split:Nee
3333         \seq_set_split:Non
3334         \l_@@_rows_seq \l_@@_end_of_row_tl { #1 }

```

We delete the last row if it is empty.

```

3335 \seq_pop_right:NN \l_@@_rows_seq \l_tmpa_tl
3336 \tl_if_empty:NF \l_tmpa_tl
3337 { \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.

```

3338 \int_compare:nNnT \l_@@_last_row_int = { -1 }
3339 { \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`).

```

3340 \tl_build_begin:N \l_@@_new_body_tl
3341 \int_zero_new:N \l_@@_nb_cols_int

```

First, we treat the first row.

```

3342 \seq_pop_left:NN \l_@@_rows_seq \l_tmpa_tl
3343 \@@_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).

```

3344 \seq_map_inline:Nn \l_@@_rows_seq
3345 {
3346   \tl_build_put_right:Nn \l_@@_new_body_tl { \\ }
3347   \@@_line_with_light_syntax:n { ##1 }
3348 }
3349 \tl_build_end:N \l_@@_new_body_tl
3350 \int_compare:nNnT \l_@@_last_col_int = { -1 }
3351 {
3352   \int_set:Nn \l_@@_last_col_int
3353   { \l_@@_nb_cols_int - 1 + \l_@@_first_col_int }
3354 }

```

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.

```

3355 \@@_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`).

```

3356 \@@_array:o \g_@@_array_preamble_tl \l_@@_new_body_tl
3357 }
3358 \cs_generate_variant:Nn \@@_line_with_light_syntax:n { o }
3359 \cs_new_protected:Npn \@@_line_with_light_syntax:n #1
3360 {
3361   \seq_clear_new:N \l_@@_cells_seq
3362   \seq_set_split:Nnn \l_@@_cells_seq { ~ } { #1 }
3363   \int_set:Nn \l_@@_nb_cols_int
3364   {
3365     \int_max:nn
3366     \l_@@_nb_cols_int
3367     { \seq_count:N \l_@@_cells_seq }
3368   }
3369   \seq_pop_left:NN \l_@@_cells_seq \l_tmpa_tl
3370   \tl_build_put_right:No \l_@@_new_body_tl \l_tmpa_tl
3371   \seq_map_inline:Nn \l_@@_cells_seq
3372   { \tl_build_put_right:Nn \l_@@_new_body_tl { & ##1 } }
3373 }

```

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.

```

3374 \cs_new_protected:Npn \@@_analyze_end:Nn #1 #2
3375 {
3376   \str_if_eq:onT \g_@@_name_env_str { #2 }
3377   { \@@_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.

```

3378   \end { #2 }
3379 }

```

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).

```

3380 \cs_new:Npn \@@_create_col_nodes:
3381 {
3382   \crrr
3383   \int_if_zero:nT \l_@@_first_col_int
3384   {
3385     \omit
3386     \hbox_overlap_left:n
3387     {
3388       \bool_if:NT \l_@@_code_before_bool
3389       { \pgfsys@markposition { \@@_env: - col - 0 } }
3390       \pgfpicture
3391       \pgfrememberpicturepositiononpagetrue
3392       \pgfcoordinate { \@@_env: - col - 0 } \pgfpointorigin
3393       \str_if_empty:NF \l_@@_name_str
3394       { \pgfnodealias { \l_@@_name_str - col - 0 } { \@@_env: - col - 0 } }
3395       \endpgfpicture
3396       \skip_horizontal:N 2\col@sep
3397       \skip_horizontal:N \g_@@_width_first_col_dim
3398     }
3399     &
3400   }
3401   \omit

```

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

```

3402   \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).

```

3403   \int_if_zero:nTF \l_@@_first_col_int
3404   {
3405     \bool_if:NT \l_@@_code_before_bool
3406     {
3407       \hbox
3408       {
3409         \skip_horizontal:N -0.5\arrayrulewidth
3410         \pgfsys@markposition { \@@_env: - col - 1 }
3411         \skip_horizontal:N 0.5\arrayrulewidth
3412       }
3413     }
3414     \pgfpicture
3415     \pgfrememberpicturepositiononpagetrue
3416     \pgfcoordinate { \@@_env: - col - 1 }
3417     { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3418     \str_if_empty:NF \l_@@_name_str
3419     { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
3420     \endpgfpicture
3421   }

```

```

3422 {
3423   \bool_if:NT \l_@@_code_before_bool
3424   {
3425     \hbox
3426     {
3427       \skip_horizontal:N 0.5\arrayrulewidth
3428       \pgfsys@markposition { \@@_env: - col - 1 }
3429       \skip_horizontal:N -0.5\arrayrulewidth
3430     }
3431   }
3432   \pgfpicture
3433   \pgfrememberpicturepositiononpagetrue
3434   \pgfcoordinate { \@@_env: - col - 1 }
3435   { \pgfpoint { 0.5 \arrayrulewidth } \c_zero_dim }
3436   \str_if_empty:NF \l_@@_name_str
3437   { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
3438   \endpgfpicture
3439 }

```

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.

```

3440 \skip_gset:Nn \g_tmpa_skip { 0 pt~plus 1 fill }
3441 \bool_if:NF \l_@@_auto_columns_width_bool
3442 { \dim_compare:nNnT \l_@@_columns_width_dim > \c_zero_dim }
3443 {
3444   \bool_lazy_and:nnTF
3445   \l_@@_auto_columns_width_bool
3446   { \bool_not_p:n \l_@@_block_auto_columns_width_bool }
3447   { \skip_gadd:Nn \g_tmpa_skip \g_@@_max_cell_width_dim }
3448   { \skip_gadd:Nn \g_tmpa_skip \l_@@_columns_width_dim }
3449   \skip_gadd:Nn \g_tmpa_skip { 2 \col@sep }
3450 }
3451 \skip_horizontal:N \g_tmpa_skip
3452 \hbox
3453 {
3454   \bool_if:NT \l_@@_code_before_bool
3455   {
3456     \hbox
3457     {
3458       \skip_horizontal:N -0.5\arrayrulewidth
3459       \pgfsys@markposition { \@@_env: - col - 2 }
3460       \skip_horizontal:N 0.5\arrayrulewidth
3461     }
3462   }
3463   \pgfpicture
3464   \pgfrememberpicturepositiononpagetrue
3465   \pgfcoordinate { \@@_env: - col - 2 }
3466   { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3467   \str_if_empty:NF \l_@@_name_str
3468   { \pgfnodealias { \l_@@_name_str - col - 2 } { \@@_env: - col - 2 } }
3469   \endpgfpicture
3470 }

```

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.

```

3471 \int_gset_eq:NN \g_tmpa_int \c_one_int
3472 \bool_if:NTF \g_@@_last_col_found_bool
3473 { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 3 } \c_zero_int } }
3474 { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 2 } \c_zero_int } }
3475 {

```

```

3476      &
3477      \omit
3478      \int_gincr:N \g_tmpa_int

```

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

```

3479      \skip_horizontal:N \g_tmpa_skip
3480      \bool_if:NT \l_@@_code_before_bool
3481      {
3482        \hbox
3483        {
3484          \skip_horizontal:N -0.5\arrayrulewidth
3485          \pgfsys@markposition
3486          { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3487          \skip_horizontal:N 0.5\arrayrulewidth
3488        }
3489      }

```

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

```

3490      \pgfpicture
3491      \pgfrememberpicturepositiononpagetrue
3492      \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3493      { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3494      \str_if_empty:NF \l_@@_name_str
3495      {
3496        \pgfnodealias
3497        { \l_@@_name_str - col - \int_eval:n { \g_tmpa_int + 1 } }
3498        { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3499      }
3500      \endpgfpicture
3501    }

```

```

3502      &
3503      \omit

```

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

```

3504      \int_if_zero:nT \g_@@_col_total_int
3505      { \skip_gset:Nn \g_tmpa_skip { 0 pt+plus 1 fill } }
3506      \skip_horizontal:N \g_tmpa_skip
3507      \int_gincr:N \g_tmpa_int
3508      \bool_lazy_any:nF
3509      {
3510        \g_@@_delims_bool
3511        \l_@@_tabular_bool
3512        { ! \clist_if_empty_p:N \l_@@_vlines_clist }
3513        \l_@@_exterior_arraycolsep_bool
3514        \l_@@_bar_at_end_of_pream_bool
3515      }
3516      { \skip_horizontal:N -\col@sep }
3517      \bool_if:NT \l_@@_code_before_bool
3518      {
3519        \hbox
3520        {
3521          \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.

```

3522          \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3523          { \skip_horizontal:N -\arraycolsep }
3524          \pgfsys@markposition
3525          { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3526          \skip_horizontal:N 0.5\arrayrulewidth
3527          \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool

```



```

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

3550 \bool_if:NT \g_@@_last_col_found_bool
3551 {
3552     \hbox_overlap_right:n
3553     {
3554         \skip_horizontal:N \g_@@_width_last_col_dim
3555         \skip_horizontal:N \col@sep
3556         \bool_if:NT \l_@@_code_before_bool
3557         {
3558             \pgfsys@markposition
3559             { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3560         }
3561         \pgfpicture
3562         \pgfrememberpicturepositiononpagetrue
3563         \pgfcoordinate
3564         { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3565         \pgfpointorigin
3566         \str_if_empty:NF \l_@@_name_str
3567         {
3568             \pgfnodealias
3569             {
3570                 \l_@@_name_str - col
3571                 - \int_eval:n { \g_@@_col_total_int + 1 }
3572             }
3573             { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3574         }
3575         \endpgfpicture
3576     }
3577 }
3578 % \cr
3579 }

```

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

```

3580 \tl_const:Nn \c_@@_preamble_first_col_tl
3581 {
3582     >
3583     {

```

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).

```

3584 \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
3585 \bool_gset_true:N \g_@@_after_col_zero_bool
3586 \@@_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.

```

3587 \hbox_set:Nw \l_@@_cell_box
3588 \@@_math_toggle:
3589 \@@_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”.

```

3590 \int_compare:nNnT \c@iRow > \c_zero_int
3591 {
3592   \bool_lazy_or:nnT
3593     { \int_compare_p:nNn \l_@@_last_row_int < \c_zero_int }
3594     { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
3595   {
3596     \l_@@_code_for_first_col_tl
3597     \xglobal \colorlet { nicematrix-first-col } { . }
3598   }
3599 }
3600 }

```

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`.

```

3601 l
3602 <
3603 {
3604   \@@_math_toggle:
3605   \hbox_set_end:
3606   \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
3607   \@@_adjust_size_box:
3608   \@@_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.

```

3609 \dim_gset:Nn \g_@@_width_first_col_dim
3610 { \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.

```

3611 \hbox_overlap_left:n
3612 {
3613   \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > \c_zero_dim
3614     \@@_node_for_cell:
3615     { \box_use_drop:N \l_@@_cell_box }
3616     \skip_horizontal:N \l_@@_left_delim_dim
3617     \skip_horizontal:N \l_@@_left_margin_dim
3618     \skip_horizontal:N \l_@@_extra_left_margin_dim
3619   }
3620   \bool_gset_false:N \g_@@_empty_cell_bool
3621   \skip_horizontal:N -2\col@sep
3622 }
3623 }

```

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

```

3624 \tl_const:Nn \c_@@_preamble_last_col_tl
3625 {
3626   >
3627   {
3628     \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).

```

3629 \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.

```

3630 \bool_gset_true:N \g_@@_last_col_found_bool
3631 \int_gincr:N \c@jCol
3632 \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.

```

3633 \hbox_set:Nw \l_@@_cell_box
3634 \@@_math_toggle:
3635 \@@_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”.

```

3636 \int_compare:nNnT \c@iRow > \c_zero_int
3637 {
3638   \bool_lazy_or:nnT
3639   { \int_compare_p:nNn \l_@@_last_row_int < \c_zero_int }
3640   { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
3641   {
3642     \l_@@_code_for_last_col_tl
3643     \xglobal \colorlet { nicematrix-last-col } { . }
3644   }
3645 }
3646 }
3647 1
3648 <
3649 {
3650   \@@_math_toggle:
3651   \hbox_set_end:
3652   \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
3653   \@@_adjust_size_box:
3654   \@@_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.

```

3655 \dim_gset:Nn \g_@@_width_last_col_dim
3656 { \dim_max:nn \g_@@_width_last_col_dim { \box_wd:N \l_@@_cell_box } }
3657 \skip_horizontal:N -2\col@sep

```

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

```

3658 \hbox_overlap_right:n
3659 {
3660   \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \c_zero_dim
3661   {
3662     \skip_horizontal:N \l_@@_right_delim_dim
3663     \skip_horizontal:N \l_@@_right_margin_dim
3664     \skip_horizontal:N \l_@@_extra_right_margin_dim
3665     \@@_node_for_cell:
3666   }
3667 }
3668 \bool_gset_false:N \g_@@_empty_cell_bool
3669 }
3670 }

```

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

```

3671 \NewDocumentEnvironment { NiceArray } { }
3672 {
3673   \bool_gset_false:N \g_@@_delims_bool
3674   \str_if_empty:NT \g_@@_name_env_str
3675   { \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).

```

3676 \NiceArrayWithDelims . .
3677 }
3678 { \endNiceArrayWithDelims }

```

We create the variants of the environment {NiceArrayWithDelims}.

```

3679 \cs_new_protected:Npn \@@_def_env:nnn #1 #2 #3
3680 {
3681   \NewDocumentEnvironment { #1 NiceArray } { }
3682   {
3683     \bool_gset_true:N \g_@@_delims_bool
3684     \str_if_empty:NT \g_@@_name_env_str
3685     { \str_gset:Nn \g_@@_name_env_str { #1 NiceArray } }
3686     \@@_test_if_math_mode:
3687     \NiceArrayWithDelims #2 #3
3688   }
3689   { \endNiceArrayWithDelims }
3690 }
3691 \@@_def_env:nnn p ( )
3692 \@@_def_env:nnn b [ ]
3693 \@@_def_env:nnn B \{ \}
3694 \@@_def_env:nnn v | |
3695 \@@_def_env:nnn V \l \r

```

14 The environment {NiceMatrix} and its variants

```

3696 \cs_generate_variant:Nn \@@_begin_of_NiceMatrix:nn { n o }
3697 \cs_new_protected:Npn \@@_begin_of_NiceMatrix:nn #1 #2
3698 {
3699   \bool_set_false:N \l_@@_preamble_bool
3700   \tl_clear:N \l_tmpa_tl
3701   \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3702   { \tl_set:Nn \l_tmpa_tl { @ { } } }
3703   \tl_put_right:Nn \l_tmpa_tl
3704   {
3705     *
3706     {
3707       \int_case:nnF \l_@@_last_col_int
3708       {
3709         { -2 } { \c@MaxMatrixCols }
3710         { -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).

```

3711       }
3712       { \int_eval:n { \l_@@_last_col_int - 1 } }
3713     }
3714     { #2 }
3715   }
3716   \tl_set:Nn \l_tmpb_tl { \use:c { #1 NiceArray } }
3717   \exp_args:No \l_tmpb_tl \l_tmpa_tl
3718 }
3719 \clist_map_inline:nn { p , b , B , v , V }
3720 {
3721   \NewDocumentEnvironment { #1 NiceMatrix } { ! 0 { } }
3722   {
3723     \bool_gset_true:N \g_@@_delims_bool

```

```

3724 \str_gset:Nn \g_@@_name_env_str { #1 NiceMatrix }
3725 \int_if_zero:nT \l_@@_last_col_int
3726 {
3727     \bool_set_true:N \l_@@_last_col_without_value_bool
3728     \int_set:Nn \l_@@_last_col_int { -1 }
3729 }
3730 \keys_set:nn { nicematrix / NiceMatrix } { ##1 }
3731 \@@_begin_of_NiceMatrix:no { #1 } \l_@@_columns_type_tl
3732 }
3733 { \use:c { end #1 NiceArray } }
3734 }

```

We define also an environment {NiceMatrix}

```

3735 \NewDocumentEnvironment { NiceMatrix } { ! 0 { } }
3736 {
3737     \str_gset:Nn \g_@@_name_env_str { NiceMatrix }
3738     \int_if_zero:nT \l_@@_last_col_int
3739     {
3740         \bool_set_true:N \l_@@_last_col_without_value_bool
3741         \int_set:Nn \l_@@_last_col_int { -1 }
3742     }
3743     \keys_set:nn { nicematrix / NiceMatrix } { #1 }
3744     \bool_lazy_or:nnT
3745     { \clist_if_empty_p:N \l_@@_vlines_clist }
3746     { \l_@@_except_borders_bool }
3747     { \bool_set_true:N \l_@@_NiceMatrix_without_vlines_bool }
3748     \@@_begin_of_NiceMatrix:no { } \l_@@_columns_type_tl
3749 }
3750 { \endNiceArray }

```

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

```

3751 \cs_new_protected:Npn \@@_NotEmpty:
3752 { \bool_gset_true:N \g_@@_not_empty_cell_bool }

```

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

```

3753 \NewDocumentEnvironment { NiceTabular } { 0 { } m ! 0 { } }
3754 {

```

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.

```

3755     \dim_compare:nNnT \l_@@_width_dim = \c_zero_dim
3756     { \dim_set_eq:NN \l_@@_width_dim \linewidth }
3757     \str_gset:Nn \g_@@_name_env_str { NiceTabular }
3758     \keys_set:nn { nicematrix / NiceTabular } { #1 , #3 }
3759     \tl_if_empty:NF \l_@@_short_caption_tl
3760     {
3761         \tl_if_empty:NT \l_@@_caption_tl
3762         {
3763             \@@_error_or_warning:n { short-caption-without~caption }
3764             \tl_set_eq:NN \l_@@_caption_tl \l_@@_short_caption_tl
3765         }
3766     }
3767     \tl_if_empty:NF \l_@@_label_tl
3768     {
3769         \tl_if_empty:NT \l_@@_caption_tl
3770         { \@@_error_or_warning:n { label~without~caption } }
3771     }
3772     \NewDocumentEnvironment { TabularNote } { b }
3773     {
3774         \bool_if:NTF \l_@@_in_code_after_bool

```

```

3775     { \@@_error_or_warning:n { TabularNote~in~CodeAfter } }
3776     {
3777         \tl_if_empty:NF \g_@@_tabularnote_tl
3778         { \tl_gput_right:Nn \g_@@_tabularnote_tl { \par } }
3779         \tl_gput_right:Nn \g_@@_tabularnote_tl { ##1 }
3780     }
3781 }
3782 { }
3783 \@@_settings_for_tabular:
3784 \NiceArray { #2 }
3785 }
3786 {
3787     \endNiceArray
3788     \bool_if:NT \c_@@_testphase_table_bool
3789     { \UseTaggingSocket { tbl / hmode / end } }
3790 }
3791 \cs_new_protected:Npn \@@_settings_for_tabular:
3792 {
3793     \bool_set_true:N \l_@@_tabular_bool
3794     \cs_set_eq:NN \@@_math_toggle: \prg_do_nothing:
3795     \cs_set_eq:NN \@@_tuning_not_tabular_begin: \prg_do_nothing:
3796     \cs_set_eq:NN \@@_tuning_not_tabular_end: \prg_do_nothing:
3797 }

3798 \NewDocumentEnvironment { NiceTabularX } { m O { } m ! O { } }
3799 {
3800     \str_gset:Nn \g_@@_name_env_str { NiceTabularX }
3801     \dim_zero_new:N \l_@@_width_dim
3802     \dim_set:Nn \l_@@_width_dim { #1 }
3803     \keys_set:nn { nicematrix / NiceTabular } { #2 , #4 }
3804     \@@_settings_for_tabular:
3805     \NiceArray { #3 }
3806 }
3807 {
3808     \endNiceArray
3809     \int_if_zero:nT \g_@@_total_X_weight_int
3810     { \@@_error:n { NiceTabularX~without~X } }
3811 }

3812 \NewDocumentEnvironment { NiceTabular* } { m O { } m ! O { } }
3813 {
3814     \str_gset:Nn \g_@@_name_env_str { NiceTabular* }
3815     \dim_set:Nn \l_@@_tabular_width_dim { #1 }
3816     \keys_set:nn { nicematrix / NiceTabular } { #2 , #4 }
3817     \@@_settings_for_tabular:
3818     \NiceArray { #3 }
3819 }
3820 { \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`).

```

3821 \cs_new_protected:Npn \@@_deal_with_rounded_corners:
3822 {
3823     \bool_lazy_all:nT
3824     {
3825         { \int_compare_p:nNn \l_@@_tab_rounded_corners_dim > \c_zero_dim }

```

```

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

```

```

3846 \cs_new_protected:Npn \@@_after_array:
3847 {

```

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`.

```

3848 \hook_gremove_code:nn { env / tabular / begin } { nicematrix }
3849 \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.

```

3850 \bool_if:NT \g_@@_last_col_found_bool
3851 { \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`.

```

3852 \bool_if:NT \l_@@_last_col_without_value_bool
3853 { \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.

```

3854 \bool_if:NT \l_@@_last_row_without_value_bool
3855 { \int_set_eq:NN \l_@@_last_row_int \g_@@_row_total_int }

```

```

3856 \tl_gput_right:Ne \g_@@_aux_tl
3857 {
3858 \seq_gset_from_clist:Nn \exp_not:N \g_@@_size_seq
3859 {
3860 \int_use:N \l_@@_first_row_int ,
3861 \int_use:N \c@iRow ,
3862 \int_use:N \g_@@_row_total_int ,
3863 \int_use:N \l_@@_first_col_int ,
3864 \int_use:N \c@jCol ,
3865 \int_use:N \g_@@_col_total_int
3866 }
3867 }

```

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`).

```

3868 \seq_if_empty:NF \g_@@_pos_of_blocks_seq
3869 {
3870   \tl_gput_right:Ne \g_@@_aux_tl
3871   {
3872     \seq_gset_from_clist:Nn \exp_not:N \g_@@_pos_of_blocks_seq
3873     { \seq_use:Nnnn \g_@@_pos_of_blocks_seq , , , }
3874   }
3875 }
3876 \seq_if_empty:NF \g_@@_multicolumn_cells_seq
3877 {
3878   \tl_gput_right:Ne \g_@@_aux_tl
3879   {
3880     \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_cells_seq
3881     { \seq_use:Nnnn \g_@@_multicolumn_cells_seq , , , }
3882     \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_sizes_seq
3883     { \seq_use:Nnnn \g_@@_multicolumn_sizes_seq , , , }
3884   }
3885 }

```

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

```

3886 \@@_create_diag_nodes:

```

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

```

3887 \pgfpicture
3888 \int_step_inline:nn \c@iRow
3889 {
3890   \pgfnodealias
3891   { \@@_env: - ##1 - last }
3892   { \@@_env: - ##1 - \int_use:N \c@jCol }
3893 }
3894 \int_step_inline:nn \c@jCol
3895 {
3896   \pgfnodealias
3897   { \@@_env: - last - ##1 }
3898   { \@@_env: - \int_use:N \c@iRow - ##1 }
3899 }
3900 \str_if_empty:NF \l_@@_name_str
3901 {
3902   \int_step_inline:nn \c@iRow
3903   {
3904     \pgfnodealias
3905     { \l_@@_name_str - ##1 - last }
3906     { \@@_env: - ##1 - \int_use:N \c@jCol }
3907   }
3908   \int_step_inline:nn \c@jCol
3909   {
3910     \pgfnodealias
3911     { \l_@@_name_str - last - ##1 }
3912     { \@@_env: - \int_use:N \c@iRow - ##1 }
3913   }
3914 }
3915 \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.

```

3916 \bool_if:NT \l_@@_parallelize_diags_bool

```

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


```

3917 {
3918   \int_gzero_new:N \g_@@_ddots_int
3919   \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.

```

3920   \dim_gzero_new:N \g_@@_delta_x_one_dim
3921   \dim_gzero_new:N \g_@@_delta_y_one_dim
3922   \dim_gzero_new:N \g_@@_delta_x_two_dim
3923   \dim_gzero_new:N \g_@@_delta_y_two_dim
3924 }

```

```

3925 \int_zero_new:N \l_@@_initial_i_int
3926 \int_zero_new:N \l_@@_initial_j_int
3927 \int_zero_new:N \l_@@_final_i_int
3928 \int_zero_new:N \l_@@_final_j_int
3929 \bool_set_false:N \l_@@_initial_open_bool
3930 \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.

```

3931 \bool_if:NT \l_@@_small_bool
3932 {
3933   \dim_set:Nn \l_@@_xdots_radius_dim { 0.7 \l_@@_xdots_radius_dim }
3934   \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.

```

3935   \dim_set:Nn \l_@@_xdots_shorten_start_dim
3936     { 0.6 \l_@@_xdots_shorten_start_dim }
3937   \dim_set:Nn \l_@@_xdots_shorten_end_dim
3938     { 0.6 \l_@@_xdots_shorten_end_dim }
3939 }

```

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

```

3940 \@@_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.

```

3941 \@@_compute_corners:

```

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

```

3942 \@@_adjust_pos_of_blocks_seq:
3943 \@@_deal_with_rounded_corners:
3944 \tl_if_empty:NF \l_@@_hlines_clist \@@_draw_hlines:
3945 \tl_if_empty:NF \l_@@_vlines_clist \@@_draw_vlines:

```

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

```

3946 \IfPackageLoadedT { tikz }
3947 {
3948   \tikzset
3949   {
3950     every~picture / .style =
3951     {
3952       overlay ,
3953       remember~picture ,

```

```

3954         name-prefix = \@@_env: -
3955     }
3956 }
3957 }
3958 \bool_if:NT \c_@@_tagging_array_bool
3959 { \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign: }
3960 \cs_set_eq:NN \SubMatrix \@@_SubMatrix
3961 \cs_set_eq:NN \UnderBrace \@@_UnderBrace
3962 \cs_set_eq:NN \OverBrace \@@_OverBrace
3963 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
3964 \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
3965 \cs_set_eq:NN \line \@@_line
3966 \g_@@_pre_code_after_tl
3967 \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.

```

3968 \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).

```

3969 \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`).

```

3970 % \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
3971 % { \@@_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:`.

```

3972 \bool_set_true:N \l_@@_in_code_after_bool
3973 \exp_last_unbraced:No \@@_CodeAfter_keys: \g_nicematrix_code_after_tl
3974 \scan_stop:
3975 \tl_gclear:N \g_nicematrix_code_after_tl
3976 \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.

```

3977 \seq_if_empty:NF \g_@@_rowlistcolors_seq { \@@_clear_rowlistcolors_seq: }
3978 \tl_if_empty:NF \g_@@_pre_code_before_tl
3979 {
3980     \tl_gput_right:Ne \g_@@_aux_tl
3981     {
3982         \tl_gset:Nn \exp_not:N \g_@@_pre_code_before_tl
3983         { \exp_not:o \g_@@_pre_code_before_tl }
3984     }
3985     \tl_gclear:N \g_@@_pre_code_before_tl
3986 }
3987 \tl_if_empty:NF \g_nicematrix_code_before_tl
3988 {
3989     \tl_gput_right:Ne \g_@@_aux_tl
3990     {
3991         \tl_gset:Nn \exp_not:N \g_@@_code_before_tl
3992         { \exp_not:o \g_nicematrix_code_before_tl }
3993     }
3994     \tl_gclear:N \g_nicematrix_code_before_tl
3995 }

3996 \str_gclear:N \g_@@_name_env_str
3997 \@@_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`.

```
3998   \cs_gset_eq:NN \CT@arc@ \@@_old_CT@arc@
3999 }
```

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`.

```
4000 \NewDocumentCommand \@@_CodeAfter_keys: { 0 { } }
4001 { \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).

```
4002 \cs_new_protected:Npn \@@_adjust_pos_of_blocks_seq:
4003 {
4004   \seq_gset_map_x:NNn \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq
4005   { \@@_adjust_pos_of_blocks_seq_i:nnnnn ##1 }
4006 }
```

The following command must *not* be protected.

```
4007 \cs_new:Npn \@@_adjust_pos_of_blocks_seq_i:nnnnn #1 #2 #3 #4 #5
4008 {
4009   { #1 }
4010   { #2 }
4011   {
4012     \int_compare:nNnTF { #3 } > { 99 }
4013     { \int_use:N \c@iRow }
4014     { #3 }
4015   }
4016   {
4017     \int_compare:nNnTF { #4 } > { 99 }
4018     { \int_use:N \c@jCol }
4019     { #4 }
4020   }
4021   { #5 }
4022 }
```

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).

```
4023 \hook_gput_code:nnn { begindocument } { . }
4024 {
4025   \cs_new_protected:Npe \@@_draw_dotted_lines:
4026   {
4027     \c_@@_pgfortikzpicture_tl
4028     \@@_draw_dotted_lines_i:
4029     \c_@@_endpgfortikzpicture_tl
4030   }
4031 }
```

¹²e.g. `\color[rgb]{0.5,0.5,0}`

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

```

4032 \cs_new_protected:Npn \@@_draw_dotted_lines_i:
4033 {
4034   \pgfrememberpicturepositiononpagetrue
4035   \pgf@relevantforpicturesizefalse
4036   \g_@@_HVdotsfor_lines_tl
4037   \g_@@_Vdots_lines_tl
4038   \g_@@_Ddots_lines_tl
4039   \g_@@_Iddots_lines_tl
4040   \g_@@_Cdots_lines_tl
4041   \g_@@_Ldots_lines_tl
4042 }

4043 \cs_new_protected:Npn \@@_restore_iRow_jCol:
4044 {
4045   \cs_if_exist:NT \theiRow { \int_gset_eq:NN \c@iRow \l_@@_old_iRow_int }
4046   \cs_if_exist:NT \thejCol { \int_gset_eq:NN \c@jCol \l_@@_old_jCol_int }
4047 }

```

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

```

4048 \pgfdeclareshape { @@_diag_node }
4049 {
4050   \savedanchor { \five }
4051   {
4052     \dim_gset_eq:NN \pgf@x \l_tmpa_dim
4053     \dim_gset_eq:NN \pgf@y \l_tmpb_dim
4054   }
4055   \anchor { 5 } { \five }
4056   \anchor { center } { \pgfpointorigin }
4057   \anchor { 1 } { \five \pgf@x = 0.2 \pgf@x \pgf@y = 0.2 \pgf@y }
4058   \anchor { 2 } { \five \pgf@x = 0.4 \pgf@x \pgf@y = 0.4 \pgf@y }
4059   \anchor { 3 } { \five \pgf@x = 0.6 \pgf@x \pgf@y = 0.6 \pgf@y }
4060   \anchor { 4 } { \five \pgf@x = 0.8 \pgf@x \pgf@y = 0.8 \pgf@y }
4061   \anchor { 6 } { \five \pgf@x = 1.2 \pgf@x \pgf@y = 1.2 \pgf@y }
4062   \anchor { 7 } { \five \pgf@x = 1.4 \pgf@x \pgf@y = 1.4 \pgf@y }
4063   \anchor { 8 } { \five \pgf@x = 1.6 \pgf@x \pgf@y = 1.6 \pgf@y }
4064   \anchor { 9 } { \five \pgf@x = 1.8 \pgf@x \pgf@y = 1.8 \pgf@y }
4065 }

```

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).

```

4066 \cs_new_protected:Npn \@@_create_diag_nodes:
4067 {
4068   \pgfpicture
4069   \pgfrememberpicturepositiononpagetrue
4070   \int_step_inline:nn { \int_max:nn \c@iRow \c@jCol }
4071   {
4072     \@@_qpoint:n { col - \int_min:nn { ##1 } { \c@jCol + 1 } }
4073     \dim_set_eq:NN \l_tmpa_dim \pgf@x
4074     \@@_qpoint:n { row - \int_min:nn { ##1 } { \c@iRow + 1 } }
4075     \dim_set_eq:NN \l_tmpb_dim \pgf@y
4076     \@@_qpoint:n { col - \int_min:nn { ##1 + 1 } { \c@jCol + 1 } }
4077     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
4078     \@@_qpoint:n { row - \int_min:nn { ##1 + 1 } { \c@iRow + 1 } }
4079     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
4080     \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.

```

4081 \dim_set:Nn \l_tmpa_dim { ( \l_@@_tmpc_dim - \l_tmpa_dim ) / 2 }
4082 \dim_set:Nn \l_tmpb_dim { ( \l_@@_tmpd_dim - \l_tmpb_dim ) / 2 }
4083 \pgfnode { @@_diag_node } { center } { } { \@@_env: - ##1 } { }
4084 \str_if_empty:NF \l_@@_name_str
4085 { \pgfnodealias { \l_@@_name_str - ##1 } { \@@_env: - ##1 } }
4086 }

```

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

```

4087 \int_set:Nn \l_tmpa_int { \int_max:nn \c@iRow \c@jCol + 1 }
4088 \@@_qpoint:n { row - \int_min:nn { \l_tmpa_int } { \c@iRow + 1 } }
4089 \dim_set_eq:NN \l_tmpa_dim \pgf@y
4090 \@@_qpoint:n { col - \int_min:nn { \l_tmpa_int } { \c@jCol + 1 } }
4091 \pgfcoordinate
4092 { \@@_env: - \int_use:N \l_tmpa_int } { \pgfpoint \pgf@x \l_tmpa_dim }
4093 \pgfnodealias
4094 { \@@_env: - last }
4095 { \@@_env: - \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
4096 \str_if_empty:NF \l_@@_name_str
4097 {
4098   \pgfnodealias
4099   { \l_@@_name_str - \int_use:N \l_tmpa_int }
4100   { \@@_env: - \int_use:N \l_tmpa_int }
4101   \pgfnodealias
4102   { \l_@@_name_str - last }
4103   { \@@_env: - last }
4104 }
4105 \endpgfpicture
4106 }

```

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.

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

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.

```

4107 \cs_new_protected:Npn \@@_find_extremities_of_line:nnnn #1 #2 #3 #4
4108 {

```

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

```

4109 \cs_set:cpn { @@ _ dotted _ #1 - #2 } { }

```

Initialization of variables.

```

4110 \int_set:Nn \l_@@_initial_i_int { #1 }
4111 \int_set:Nn \l_@@_initial_j_int { #2 }
4112 \int_set:Nn \l_@@_final_i_int { #1 }
4113 \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.

```

4114 \bool_set_false:N \l_@@_stop_loop_bool
4115 \bool_do_until:Nn \l_@@_stop_loop_bool
4116 {
4117   \int_add:Nn \l_@@_final_i_int { #3 }
4118   \int_add:Nn \l_@@_final_j_int { #4 }
4119   \bool_set_false:N \l_@@_final_open_bool

```

We test if we are still in the matrix. Since this is the core of the loop, we **optimize** the code by using a TeX-style of conditionals.

```

4120   \if_int_compare:w \l_@@_final_i_int > \l_@@_row_max_int
4121     \if_int_compare:w #3 = \c_one_int
4122       \bool_set_true:N \l_@@_final_open_bool
4123     \else:
4124       \if_int_compare:w \l_@@_final_j_int > \l_@@_col_max_int
4125         \bool_set_true:N \l_@@_final_open_bool
4126       \fi:
4127     \fi:
4128   \else:
4129     \if_int_compare:w \l_@@_final_j_int < \l_@@_col_min_int
4130       \if_int_compare:w #4 = -1
4131         \bool_set_true:N \l_@@_final_open_bool
4132       \fi:
4133     \else:
4134       \if_int_compare:w \l_@@_final_j_int > \l_@@_col_max_int
4135         \if_int_compare:w #4 = \c_one_int
4136           \bool_set_true:N \l_@@_final_open_bool
4137         \fi:
4138       \fi:
4139     \fi:
4140   \fi:
4141   \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.

```

4142   {

```

We do a step backwards.

```

4143     \int_sub:Nn \l_@@_final_i_int { #3 }
4144     \int_sub:Nn \l_@@_final_j_int { #4 }
4145     \bool_set_true:N \l_@@_stop_loop_bool
4146   }

```

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`.

```

4147   {
4148     \cs_if_exist:cTF
4149     {
4150       @@ _ dotted _
4151       \int_use:N \l_@@_final_i_int -
4152       \int_use:N \l_@@_final_j_int
4153     }

```

```

4154     {
4155         \int_sub:Nn \l_@@_final_i_int { #3 }
4156         \int_sub:Nn \l_@@_final_j_int { #4 }
4157         \bool_set_true:N \l_@@_final_open_bool
4158         \bool_set_true:N \l_@@_stop_loop_bool
4159     }
4160     {
4161         \cs_if_exist:cTF
4162         {
4163             pgf @ sh @ ns @ \@@_env:
4164             - \int_use:N \l_@@_final_i_int
4165             - \int_use:N \l_@@_final_j_int
4166         }
4167         { \bool_set_true:N \l_@@_stop_loop_bool }

```

If the case is empty, we declare that the cell as non-empty. Indeed, we will draw a dotted line and the cell will be on that dotted line. All the cells of a dotted line have to be marked as “dotted” because we don’t want intersections between dotted lines. We recall that the research of the extremities of the lines are all done in the same TeX group (the group of the environment), even though, when the extremities are found, each line is drawn in a TeX group that we will open for the options of the line.

```

4168     {
4169         \cs_set:cpn
4170         {
4171             @@ _ dotted _
4172             \int_use:N \l_@@_final_i_int -
4173             \int_use:N \l_@@_final_j_int
4174         }
4175         { }
4176     }
4177 }
4178 }
4179 }

```

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

```

4180 \bool_set_false:N \l_@@_stop_loop_bool

```

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

```

4181 \int_set:Nn \l_tmpa_int { \l_@@_col_min_int - 1 }
4182 \bool_do_until:Nn \l_@@_stop_loop_bool
4183 {
4184     \int_sub:Nn \l_@@_initial_i_int { #3 }
4185     \int_sub:Nn \l_@@_initial_j_int { #4 }
4186     \bool_set_false:N \l_@@_initial_open_bool

```

We test if we are still in the matrix. Since this is the core of the loop, we **optimize** the code by using a TeX-style of conditionals.

```

4187 \if_int_compare:w \l_@@_initial_i_int < \l_@@_row_min_int
4188 \if_int_compare:w #3 = \c_one_int
4189     \bool_set_true:N \l_@@_initial_open_bool
4190 \else:

```

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

```

4191 \if_int_compare:w \l_@@_initial_j_int = \l_tmpa_int
4192     \bool_set_true:N \l_@@_initial_open_bool
4193 \fi:
4194 \fi:
4195 \else:
4196     \if_int_compare:w \l_@@_initial_j_int < \l_@@_col_min_int
4197     \if_int_compare:w #4 = \c_one_int
4198         \bool_set_true:N \l_@@_initial_open_bool
4199     \fi:
4200 \else:

```

```

4201         \if_int_compare:w \l_@@_initial_j_int > \l_@@_col_max_int
4202         \if_int_compare:w #4 = -1
4203             \bool_set_true:N \l_@@_initial_open_bool
4204         \fi:
4205     \fi:
4206 \fi:
4207 \fi:
4208 \bool_if:NTF \l_@@_initial_open_bool
4209 {
4210     \int_add:Nn \l_@@_initial_i_int { #3 }
4211     \int_add:Nn \l_@@_initial_j_int { #4 }
4212     \bool_set_true:N \l_@@_stop_loop_bool
4213 }
4214 {
4215     \cs_if_exist:cTF
4216     {
4217         @@ _ dotted _
4218         \int_use:N \l_@@_initial_i_int -
4219         \int_use:N \l_@@_initial_j_int
4220     }
4221     {
4222         \int_add:Nn \l_@@_initial_i_int { #3 }
4223         \int_add:Nn \l_@@_initial_j_int { #4 }
4224         \bool_set_true:N \l_@@_initial_open_bool
4225         \bool_set_true:N \l_@@_stop_loop_bool
4226     }
4227     {
4228         \cs_if_exist:cTF
4229         {
4230             pgf @ sh @ ns @ \@@_env:
4231             - \int_use:N \l_@@_initial_i_int
4232             - \int_use:N \l_@@_initial_j_int
4233         }
4234         { \bool_set_true:N \l_@@_stop_loop_bool }
4235         {
4236             \cs_set:cpn
4237             {
4238                 @@ _ dotted _
4239                 \int_use:N \l_@@_initial_i_int -
4240                 \int_use:N \l_@@_initial_j_int
4241             }
4242             { }
4243         }
4244     }
4245 }
4246 }

```

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.

```

4247 \seq_gput_right:Ne \g_@@_pos_of_xdots_seq
4248 {
4249     { \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`.

```

4250     { \int_min:nn \l_@@_initial_j_int \l_@@_final_j_int }
4251     { \int_use:N \l_@@_final_i_int }
4252     { \int_max:nn \l_@@_initial_j_int \l_@@_final_j_int }
4253     { } % for the name of the block
4254 }
4255 }

```

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.

```

4256 \cs_new_protected:Npn \@@_open_shorten:
4257 {
4258   \bool_if:NT \l_@@_initial_open_bool
4259     { \dim_zero:N \l_@@_xdots_shorten_start_dim }
4260   \bool_if:NT \l_@@_final_open_bool
4261     { \dim_zero:N \l_@@_xdots_shorten_end_dim }
4262 }

```

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).

```

4263 \cs_new_protected:Npn \@@_adjust_to_submatrix:nn #1 #2
4264 {
4265   \int_set_eq:NN \l_@@_row_min_int \c_one_int
4266   \int_set_eq:NN \l_@@_col_min_int \c_one_int
4267   \int_set_eq:NN \l_@@_row_max_int \c@iRow
4268   \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`.

```

4269   \seq_if_empty:NF \g_@@_submatrix_seq
4270   {
4271     \seq_map_inline:Nn \g_@@_submatrix_seq
4272       { \@@_adjust_to_submatrix:nnnnnn { #1 } { #2 } ##1 }
4273   }
4274 }

```

#1 and #2 are the numbers of row and columns of the cell where the command of dotted line (ex.: `\Vdots`) has been issued. #3, #4, #5 and #6 are the specification (in *i* and *j*) of the submatrix we are analyzing.

Here is the programming of that command with the standard syntax of L3.

```

\cs_new_protected:Npn \@@_adjust_to_submatrix:nnnnnn #1 #2 #3 #4 #5 #6
{
  \bool_if:nT
  {
    \int_compare_p:n { #3 <= #1 <= #5 }
    &&
    \int_compare_p:n { #4 <= #2 <= #6 }
  }
  {
    \int_set:Nn \l_@@_row_min_int { \int_max:nn \l_@@_row_min_int { #3 } }
    \int_set:Nn \l_@@_col_min_int { \int_max:nn \l_@@_col_min_int { #4 } }
    \int_set:Nn \l_@@_row_max_int { \int_min:nn \l_@@_row_max_int { #5 } }
    \int_set:Nn \l_@@_col_max_int { \int_min:nn \l_@@_col_max_int { #6 } }
  }
}

```

However, for efficiency, we will use the following version.

```

4275 \cs_new_protected:Npn \@@_adjust_to_submatrix:nnnnnn #1 #2 #3 #4 #5 #6
4276 {
4277   \if_int_compare:w #3 > #1
4278   \else:
4279     \if_int_compare:w #1 > #5
4280     \else:
4281       \if_int_compare:w #4 > #2

```

```

4282     \else:
4283     \if_int_compare:w #2 > #6
4284     \else:
4285     \if_int_compare:w \l_@@_row_min_int < #3 \l_@@_row_min_int = #3 \fi:
4286     \if_int_compare:w \l_@@_col_min_int < #4 \l_@@_col_min_int = #4 \fi:
4287     \if_int_compare:w \l_@@_row_max_int < #5 \l_@@_row_max_int = #5 \fi:
4288     \if_int_compare:w \l_@@_col_max_int < #6 \l_@@_col_max_int = #6 \fi:
4289     \fi:
4290     \fi:
4291     \fi:
4292     \fi:
4293 }

4294 \cs_new_protected:Npn \@@_set_initial_coords:
4295 {
4296     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4297     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4298 }
4299 \cs_new_protected:Npn \@@_set_final_coords:
4300 {
4301     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4302     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4303 }
4304 \cs_new_protected:Npn \@@_set_initial_coords_from_anchor:n #1
4305 {
4306     \pgfpointanchor
4307     {
4308         \@@_env:
4309         - \int_use:N \l_@@_initial_i_int
4310         - \int_use:N \l_@@_initial_j_int
4311     }
4312     { #1 }
4313     \@@_set_initial_coords:
4314 }
4315 \cs_new_protected:Npn \@@_set_final_coords_from_anchor:n #1
4316 {
4317     \pgfpointanchor
4318     {
4319         \@@_env:
4320         - \int_use:N \l_@@_final_i_int
4321         - \int_use:N \l_@@_final_j_int
4322     }
4323     { #1 }
4324     \@@_set_final_coords:
4325 }
4326 \cs_new_protected:Npn \@@_open_x_initial_dim:
4327 {
4328     \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
4329     \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
4330     {
4331         \cs_if_exist:cT
4332         { \pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4333         {
4334             \pgfpointanchor
4335             { \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4336             { west }
4337             \dim_set:Nn \l_@@_x_initial_dim
4338             { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
4339         }
4340     }

```

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

```

4341     \dim_compare:nNnT \l_@@_x_initial_dim = \c_max_dim

```

```

4342 {
4343   \l_@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4344   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4345   \dim_add:Nn \l_@@_x_initial_dim \col@sep
4346 }
4347 }
4348 \cs_new_protected:Npn \l_@@_open_x_final_dim:
4349 {
4350   \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
4351   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
4352   {
4353     \cs_if_exist:cT
4354     { \pgf @ sh @ ns @ \l_@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4355     {
4356       \pgfpointanchor
4357       { \l_@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4358       { east }
4359       \dim_compare:nNnT \pgf@x > \l_@@_x_final_dim
4360       { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
4361     }
4362   }
4363   \dim_compare:nNnT \l_@@_x_final_dim = { - \c_max_dim }
4364   {
4365     \l_@@_qpoint:n { col - \int_eval:n { \l_@@_final_j_int + 1 } }
4366     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4367     \dim_sub:Nn \l_@@_x_final_dim \col@sep
4368   }
4369 }

```

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

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.

```

4370 \cs_new_protected:Npn \l_@@_draw_Ldots:nnn #1 #2 #3
4371 {
4372   \l_@@_adjust_to_submatrix:nn { #1 } { #2 }
4373   \cs_if_free:cT { \l_@@_dotted _ #1 - #2 }
4374   {
4375     \l_@@_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.

```

4376   \group_begin:
4377   \l_@@_open_shorten:
4378   \int_if_zero:nTF { #1 }
4379   { \color { nicematrix-first-row } }
4380   {

```

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.

```

4381     \int_compare:nNnT { #1 } = \l_@@_last_row_int
4382     { \color { nicematrix-last-row } }
4383   }
4384   \keys_set:nn { nicematrix / xdots } { #3 }
4385   \tl_if_empty:oF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4386   \l_@@_actually_draw_Ldots:
4387   \group_end:
4388 }
4389 }

```

The command `\l_@@_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`.

```

4390 \cs_new_protected:Npn \@@_actually_draw_Ldots:
4391 {
4392   \bool_if:NTF \l_@@_initial_open_bool
4393   {
4394     \@@_open_x_initial_dim:
4395     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4396     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4397   }
4398   { \@@_set_initial_coords_from_anchor:n { base~east } }
4399   \bool_if:NTF \l_@@_final_open_bool
4400   {
4401     \@@_open_x_final_dim:
4402     \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4403     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4404   }
4405   { \@@_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.

```

4406   \bool_lazy_all:nTF
4407   {
4408     \l_@@_initial_open_bool
4409     \l_@@_final_open_bool
4410     { \int_compare_p:nNn \l_@@_initial_i_int = \l_@@_last_row_int }
4411   }
4412   {
4413     \dim_add:Nn \l_@@_y_initial_dim \c_@@_shift_Ldots_last_row_dim
4414     \dim_add:Nn \l_@@_y_final_dim \c_@@_shift_Ldots_last_row_dim
4415   }

```

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 (?).

```

4416   {
4417     \dim_add:Nn \l_@@_y_initial_dim \l_@@_xdots_radius_dim
4418     \dim_add:Nn \l_@@_y_final_dim \l_@@_xdots_radius_dim
4419   }
4420   \@@_draw_line:
4421 }

```

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.

```

4422 \cs_new_protected:Npn \@@_draw_Cdots:nnn #1 #2 #3
4423 {
4424   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4425   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4426   {
4427     \@@_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.

```

4428   \group_begin:
4429   \@@_open_shorten:

```

```

4430 \int_if_zero:nTF { #1 }
4431 { \color { nicematrix-first-row } }
4432 {

```

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.

```

4433 \int_compare:nNtT { #1 } = \l_@@_last_row_int
4434 { \color { nicematrix-last-row } }
4435 }
4436 \keys_set:nn { nicematrix / xdots } { #3 }
4437 \tl_if_empty:oF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4438 \@@_actually_draw_Cdots:
4439 \group_end:
4440 }
4441 }

```

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`.

```

4442 \cs_new_protected:Npn \@@_actually_draw_Cdots:
4443 {
4444   \bool_if:NTF \l_@@_initial_open_bool
4445   { \@@_open_x_initial_dim: }
4446   { \@@_set_initial_coords_from_anchor:n { mid-east } }
4447   \bool_if:NTF \l_@@_final_open_bool
4448   { \@@_open_x_final_dim: }
4449   { \@@_set_final_coords_from_anchor:n { mid-west } }
4450   \bool_lazy_and:nnTF
4451   \l_@@_initial_open_bool
4452   \l_@@_final_open_bool
4453   {
4454     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int }
4455     \dim_set_eq:NN \l_tmpa_dim \pgf@y
4456     \@@_qpoint:n { row - \int_eval:n { \l_@@_initial_i_int + 1 } }
4457     \dim_set:Nn \l_@@_y_initial_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
4458     \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
4459   }
4460   {
4461     \bool_if:NT \l_@@_initial_open_bool
4462     { \dim_set_eq:NN \l_@@_y_initial_dim \l_@@_y_final_dim }
4463     \bool_if:NT \l_@@_final_open_bool
4464     { \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim }
4465   }
4466   \@@_draw_line:
4467 }
4468 \cs_new_protected:Npn \@@_open_y_initial_dim:
4469 {
4470   \dim_set:Nn \l_@@_y_initial_dim { - \c_max_dim }
4471   \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
4472   {
4473     \cs_if_exist:cT
4474     { pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4475     {
4476       \pgfpointanchor

```

```

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

```

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.

```

4518 \cs_new_protected:Npn \@@_draw_Vdots:nnn #1 #2 #3
4519 {
4520     \@@_adjust_to_submatrix:nn { #1 } { #2 }
4521     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4522     {
4523         \@@_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.

```

4524     \group_begin:
4525     \@@_open_shorten:
4526     \int_if_zero:nTF { #2 }
4527     { \color { nicematrix-first-col } }
4528     {
4529         \int_compare:nNtT { #2 } = \l_@@_last_col_int
4530         { \color { nicematrix-last-col } }
4531     }
4532     \keys_set:nn { nicematrix / xdots } { #3 }
4533     \tl_if_empty:oF \l_@@_xdots_color_tl
4534     { \color { \l_@@_xdots_color_tl } }

```

```

4535         \@@_actually_draw_Vdots:
4536     \group_end:
4537 }
4538 }

```

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`.

```

4539 \cs_new_protected:Npn \@@_actually_draw_Vdots:
4540 {

```

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

```

4541     \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.

```

4542     {
4543         \@@_open_y_initial_dim:
4544         \@@_open_y_final_dim:
4545         \int_if_zero:nTF \l_@@_initial_j_int

```

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

```

4546     {
4547         \@@_qpoint:n { col - 1 }
4548         \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4549         \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
4550         \dim_sub:Nn \l_@@_x_initial_dim \l_@@_extra_left_margin_dim
4551         \dim_sub:Nn \l_@@_x_initial_dim \c_@@_shift_exterior_Vdots_dim
4552     }
4553     {
4554         \bool_lazy_and:nnTF
4555         { \int_compare_p:nNn \l_@@_last_col_int > { -2 } }
4556         { \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”.

```

4557     {
4558         \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4559         \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4560         \dim_add:Nn \l_@@_x_initial_dim \l_@@_right_margin_dim
4561         \dim_add:Nn \l_@@_x_initial_dim \l_@@_extra_right_margin_dim
4562         \dim_add:Nn \l_@@_x_initial_dim \c_@@_shift_exterior_Vdots_dim
4563     }

```

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

```

4564     {
4565         \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4566         \dim_set_eq:NN \l_tmpa_dim \pgf@x
4567         \@@_qpoint:n { col - \int_eval:n { \l_@@_initial_j_int + 1 } }
4568         \dim_set:Nn \l_@@_x_initial_dim { ( \pgf@x + \l_tmpa_dim ) / 2 }
4569     }
4570 }
4571 }

```

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 `l` or may be considered as if.

```

4572 {
4573   \bool_set_false:N \l_tmpa_bool
4574   \bool_if:NF \l_@@_initial_open_bool
4575   {
4576     \bool_if:NF \l_@@_final_open_bool
4577     {
4578       \@@_set_initial_coords_from_anchor:n { south-west }
4579       \@@_set_final_coords_from_anchor:n { north-west }
4580       \bool_set:Nn \l_tmpa_bool
4581       { \dim_compare_p:nNn \l_@@_x_initial_dim = \l_@@_x_final_dim }
4582     }
4583   }

```

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

```

4584   \bool_if:NTF \l_@@_initial_open_bool
4585   {
4586     \@@_open_y_initial_dim:
4587     \@@_set_final_coords_from_anchor:n { north }
4588     \dim_set_eq:NN \l_@@_x_initial_dim \l_@@_x_final_dim
4589   }
4590   {
4591     \@@_set_initial_coords_from_anchor:n { south }
4592     \bool_if:NTF \l_@@_final_open_bool
4593     \@@_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.

```

4594   {
4595     \@@_set_final_coords_from_anchor:n { north }
4596     \dim_compare:nNnF \l_@@_x_initial_dim = \l_@@_x_final_dim
4597     {
4598       \dim_set:Nn \l_@@_x_initial_dim
4599       {
4600         \bool_if:NTF \l_tmpa_bool \dim_min:nn \dim_max:nn
4601         \l_@@_x_initial_dim \l_@@_x_final_dim
4602       }
4603     }
4604   }
4605 }
4606 }
4607 \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
4608 \@@_draw_line:
4609 }

```

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.

```

4610 \cs_new_protected:Npn \@@_draw_Ddots:nnn #1 #2 #3
4611 {
4612   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4613   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4614   {
4615     \@@_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.

```

4616   \group_begin:
4617   \@@_open_shorten:

```



```

4618         \keys_set:nn { nicematrix / xdots } { #3 }
4619         \tl_if_empty:oF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4620         \@@_actually_draw_Ddots:
4621     \group_end:
4622 }
4623 }

```

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.`

```

4624 \cs_new_protected:Npn \@@_actually_draw_Ddots:
4625 {
4626     \bool_if:NTF \l_@@_initial_open_bool
4627     {
4628         \@@_open_y_initial_dim:
4629         \@@_open_x_initial_dim:
4630     }
4631     { \@@_set_initial_coords_from_anchor:n { south-east } }
4632     \bool_if:NTF \l_@@_final_open_bool
4633     {
4634         \@@_open_x_final_dim:
4635         \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4636     }
4637     { \@@_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.

```

4638     \bool_if:NT \l_@@_parallelize_diags_bool
4639     {
4640         \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).

```

4641         \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.

```

4642     {
4643         \dim_gset:Nn \g_@@_delta_x_one_dim
4644         { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4645         \dim_gset:Nn \g_@@_delta_y_one_dim
4646         { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4647     }

```

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`.

```

4648     {
4649         \dim_compare:nNnF \g_@@_delta_x_one_dim = \c_zero_dim
4650         {
4651             \dim_set:Nn \l_@@_y_final_dim
4652             {
4653                 \l_@@_y_initial_dim +
4654                 ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4655                 \dim_ratio:nn \g_@@_delta_y_one_dim \g_@@_delta_x_one_dim
4656             }

```

```

4657     }
4658   }
4659 }
4660 \@@_draw_line:
4661 }

```

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.

```

4662 \cs_new_protected:Npn \@@_draw_Iddots:nnn #1 #2 #3
4663 {
4664   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4665   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4666   {
4667     \@@_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.

```

4668     \group_begin:
4669     \@@_open_shorten:
4670     \keys_set:nn { nicematrix / xdots } { #3 }
4671     \tl_if_empty:oF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4672     \@@_actually_draw_Iddots:
4673   \group_end:
4674 }
4675 }

```

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.`

```

4676 \cs_new_protected:Npn \@@_actually_draw_Iddots:
4677 {
4678   \bool_if:NTF \l_@@_initial_open_bool
4679   {
4680     \@@_open_y_initial_dim:
4681     \@@_open_x_initial_dim:
4682   }
4683   { \@@_set_initial_coords_from_anchor:n { south-west } }
4684   \bool_if:NTF \l_@@_final_open_bool
4685   {
4686     \@@_open_y_final_dim:
4687     \@@_open_x_final_dim:
4688   }
4689   { \@@_set_final_coords_from_anchor:n { north-east } }
4690   \bool_if:NT \l_@@_parallelize_diags_bool
4691   {
4692     \int_gincr:N \g_@@_iddots_int
4693     \int_compare:nNnTF \g_@@_iddots_int = \c_one_int
4694     {
4695       \dim_gset:Nn \g_@@_delta_x_two_dim
4696       { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4697       \dim_gset:Nn \g_@@_delta_y_two_dim
4698       { \l_@@_y_final_dim - \l_@@_y_initial_dim }

```

```

4699     }
4700     {
4701         \dim_compare:nNnF \g_@@_delta_x_two_dim = \c_zero_dim
4702         {
4703             \dim_set:Nn \l_@@_y_final_dim
4704             {
4705                 \l_@@_y_initial_dim +
4706                 ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4707                 \dim_ratio:nn \g_@@_delta_y_two_dim \g_@@_delta_x_two_dim
4708             }
4709         }
4710     }
4711 }
4712 \@@_draw_line:
4713 }

```

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`

```

4714 \cs_new_protected:Npn \@@_draw_line:
4715 {
4716     \pgfrememberpicturepositiononpagetrue
4717     \pgf@relevantforpicturesizefalse
4718     \bool_lazy_or:nnTF
4719     { \tl_if_eq_p:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl }
4720     \l_@@_dotted_bool
4721     \@@_draw_standard_dotted_line:
4722     \@@_draw_unstandard_dotted_line:
4723 }

```

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.

```

4724 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:
4725 {
4726     \begin { scope }
4727     \@@_draw_unstandard_dotted_line:o
4728     { \l_@@_xdots_line_style_tl , \l_@@_xdots_color_tl }
4729 }

```

We have used the fact that, in PGF, a 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.

```

4730 \cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:n { o }
4731 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:n #1
4732 {

```

```

4733 \@@_draw_unstandard_dotted_line:nnoo
4734 { #1 }
4735 \l_@@_xdots_up_tl
4736 \l_@@_xdots_down_tl
4737 \l_@@_xdots_middle_tl
4738 }

```

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

```

4739 \hook_gput_code:nnn { begindocument } { . }
4740 {
4741   \IfPackageLoadedT { tikz }
4742   {
4743     \tikzset
4744     {
4745       @@_node_above / .style = { sloped , above } ,
4746       @@_node_below / .style = { sloped , below } ,
4747       @@_node_middle / .style =
4748       {
4749         sloped ,
4750         inner~sep = \c_@@_innersep_middle_dim
4751       }
4752     }
4753   }
4754 }

4755 \cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:nnnn { n o o o }
4756 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:nnnn #1 #2 #3 #4
4757 {

```

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.

```

4758 \dim_zero_new:N \l_@@_l_dim
4759 \dim_set:Nn \l_@@_l_dim
4760 {
4761   \fp_to_dim:n
4762   {
4763     sqrt
4764     (
4765       ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
4766       +
4767       ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
4768     )
4769   }
4770 }

```

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.

```

4771 \dim_compare:nNnT \l_@@_l_dim < \c_@@_max_l_dim
4772 {
4773   \dim_compare:nNnT \l_@@_l_dim > { 1 pt }
4774   \@@_draw_unstandard_dotted_line_i:
4775 }

```

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

```

4776 \bool_if:NT \l_@@_xdots_h_labels_bool
4777 {

```

```

4778 \tikzset
4779 {
4780     @@_node_above / .style = { auto = left } ,
4781     @@_node_below / .style = { auto = right } ,
4782     @@_node_middle / .style = { inner-sep = \c_@@_innersep_middle_dim }
4783 }
4784 }
4785 \tl_if_empty:nF { #4 }
4786 { \tikzset { @@_node_middle / .append~style = { fill = white } } }
4787 \draw
4788 [ #1 ]
4789 ( \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).

```

4790 -- node [ @@_node_middle ] { $ \scriptstyle #4 $ }
4791 node [ @@_node_below ] { $ \scriptstyle #3 $ }
4792 node [ @@_node_above ] { $ \scriptstyle #2 $ }
4793 ( \l_@@_x_final_dim , \l_@@_y_final_dim ) ;
4794 \end { scope }
4795 }
4796 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line_i:
4797 {
4798     \dim_set:Nn \l_tmpa_dim
4799     {
4800         \l_@@_x_initial_dim
4801         + ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
4802         * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
4803     }
4804     \dim_set:Nn \l_tmpb_dim
4805     {
4806         \l_@@_y_initial_dim
4807         + ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
4808         * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
4809     }
4810     \dim_set:Nn \l_@@_tmpc_dim
4811     {
4812         \l_@@_x_final_dim
4813         - ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
4814         * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
4815     }
4816     \dim_set:Nn \l_@@_tmpd_dim
4817     {
4818         \l_@@_y_final_dim
4819         - ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
4820         * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
4821     }
4822     \dim_set_eq:NN \l_@@_x_initial_dim \l_tmpa_dim
4823     \dim_set_eq:NN \l_@@_y_initial_dim \l_tmpb_dim
4824     \dim_set_eq:NN \l_@@_x_final_dim \l_@@_tmpc_dim
4825     \dim_set_eq:NN \l_@@_y_final_dim \l_@@_tmpd_dim
4826 }

```

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

```

4827 \cs_new_protected:Npn \@@_draw_standard_dotted_line:
4828 {
4829     \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.

```

4830     \dim_zero_new:N \l_@@_l_dim
4831     \dim_set:Nn \l_@@_l_dim

```

```

4832     {
4833         \fp_to_dim:n
4834         {
4835             sqrt
4836             (
4837                 ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
4838                 +
4839                 ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
4840             )
4841         }
4842     }

```

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.

```

4843     \dim_compare:nNnT \l_@@_l_dim < \c_@@_max_l_dim
4844     {
4845         \dim_compare:nNnT \l_@@_l_dim > { 1 pt }
4846         \@@_draw_standard_dotted_line_i:
4847     }
4848     \group_end:
4849     \bool_lazy_all:nF
4850     {
4851         { \tl_if_empty_p:N \l_@@_xdots_up_tl }
4852         { \tl_if_empty_p:N \l_@@_xdots_down_tl }
4853         { \tl_if_empty_p:N \l_@@_xdots_middle_tl }
4854     }
4855     \l_@@_labels_standard_dotted_line:
4856 }
4857 \dim_const:Nn \c_@@_max_l_dim { 50 cm }
4858 \cs_new_protected:Npn \@@_draw_standard_dotted_line_i:
4859 {

```

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

```

4860     \int_set:Nn \l_tmpa_int
4861     {
4862         \dim_ratio:nn
4863         {
4864             \l_@@_l_dim
4865             - \l_@@_xdots_shorten_start_dim
4866             - \l_@@_xdots_shorten_end_dim
4867         }
4868         \l_@@_xdots_inter_dim
4869     }

```

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

```

4870     \dim_set:Nn \l_tmpa_dim
4871     {
4872         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4873         \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
4874     }
4875     \dim_set:Nn \l_tmpb_dim
4876     {
4877         ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
4878         \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
4879     }

```

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.

```

4880     \dim_gadd:Nn \l_@@_x_initial_dim

```

```

4881 {
4882   ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4883   \dim_ratio:nn
4884   {
4885     \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
4886     + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
4887   }
4888   { 2 \l_@@_l_dim }
4889 }
4890 \dim_gadd:Nn \l_@@_y_initial_dim
4891 {
4892   ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
4893   \dim_ratio:nn
4894   {
4895     \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
4896     + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
4897   }
4898   { 2 \l_@@_l_dim }
4899 }
4900 \pgf@relevantforpicturesizefalse
4901 \int_step_inline:nnn \c_zero_int \l_tmpa_int
4902 {
4903   \pgfpathcircle
4904   { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
4905   { \l_@@_xdots_radius_dim }
4906   \dim_add:Nn \l_@@_x_initial_dim \l_tmpa_dim
4907   \dim_add:Nn \l_@@_y_initial_dim \l_tmpb_dim
4908 }
4909 \pgfusepathqfill
4910 }

4911 \cs_new_protected:Npn \l_@@_labels_standard_dotted_line:
4912 {
4913   \pgfscope
4914   \pgftransformshift
4915   {
4916     \pgfpointlineatime { 0.5 }
4917     { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
4918     { \pgfpoint \l_@@_x_final_dim \l_@@_y_final_dim }
4919   }
4920   \fp_set:Nn \l_tmpa_fp
4921   {
4922     atand
4923     (
4924       \l_@@_y_final_dim - \l_@@_y_initial_dim ,
4925       \l_@@_x_final_dim - \l_@@_x_initial_dim
4926     )
4927   }
4928   \pgftransformrotate { \fp_use:N \l_tmpa_fp }
4929   \bool_if:NF \l_@@_xdots_h_labels_bool { \fp_zero:N \l_tmpa_fp }
4930   \tl_if_empty:NF \l_@@_xdots_middle_tl
4931   {
4932     \begin { pgfscope }
4933     \pgfset { inner~sep = \c_@@_innersep_middle_dim }
4934     \pgfnode
4935     { rectangle }
4936     { center }
4937     {
4938       \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4939       {
4940         \c_math_toggle_token
4941         \scriptstyle \l_@@_xdots_middle_tl
4942         \c_math_toggle_token

```

```

4943     }
4944   }
4945   { }
4946   {
4947     \pgfsetfillcolor { white }
4948     \pgfusepath { fill }
4949   }
4950   \end { pgfscope }
4951 }
4952 \tl_if_empty:NF \l_@@_xdots_up_tl
4953 {
4954   \pgfnode
4955   { rectangle }
4956   { south }
4957   {
4958     \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4959     {
4960       \c_math_toggle_token
4961       \scriptstyle \l_@@_xdots_up_tl
4962       \c_math_toggle_token
4963     }
4964   }
4965   { }
4966   { \pgfusepath { } }
4967 }
4968 \tl_if_empty:NF \l_@@_xdots_down_tl
4969 {
4970   \pgfnode
4971   { rectangle }
4972   { north }
4973   {
4974     \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4975     {
4976       \c_math_toggle_token
4977       \scriptstyle \l_@@_xdots_down_tl
4978       \c_math_toggle_token
4979     }
4980   }
4981   { }
4982   { \pgfusepath { } }
4983 }
4984 \endpgfscope
4985 }

```

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.

```

4986 \hook_gput_code:nnn { begindocument } { . }
4987 {
4988   \cs_set_nopar:Npn \l_@@_argspec_tl { m E { _ ^ : } { { } { } { } } }
4989   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl

```



```

4990 \cs_new_protected:Npn \@@_Ldots
4991 { \@@_collect_options:n { \@@_Ldots_i } }
4992 \exp_args:NNo \NewDocumentCommand \@@_Ldots_i \l_@@_argspec_tl
4993 {
4994   \int_if_zero:nTF \c@jCol
4995   { \@@_error:nn { in~first~col } \Ldots }
4996   {
4997     \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
4998     { \@@_error:nn { in~last~col } \Ldots }
4999     {
5000       \@@_instruction_of_type:nnn \c_false_bool { Ldots }
5001       { #1 , down = #2 , up = #3 , middle = #4 }
5002     }
5003   }
5004   \bool_if:NF \l_@@_nullify_dots_bool
5005   { \phantom { \ensuremath { \@@_old_ldots } } }
5006   \bool_gset_true:N \g_@@_empty_cell_bool
5007 }

5008 \cs_new_protected:Npn \@@_Cdots
5009 { \@@_collect_options:n { \@@_Cdots_i } }
5010 \exp_args:NNo \NewDocumentCommand \@@_Cdots_i \l_@@_argspec_tl
5011 {
5012   \int_if_zero:nTF \c@jCol
5013   { \@@_error:nn { in~first~col } \Cdots }
5014   {
5015     \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
5016     { \@@_error:nn { in~last~col } \Cdots }
5017     {
5018       \@@_instruction_of_type:nnn \c_false_bool { Cdots }
5019       { #1 , down = #2 , up = #3 , middle = #4 }
5020     }
5021   }
5022   \bool_if:NF \l_@@_nullify_dots_bool
5023   { \phantom { \ensuremath { \@@_old_cdots } } }
5024   \bool_gset_true:N \g_@@_empty_cell_bool
5025 }

5026 \cs_new_protected:Npn \@@_Vdots
5027 { \@@_collect_options:n { \@@_Vdots_i } }
5028 \exp_args:NNo \NewDocumentCommand \@@_Vdots_i \l_@@_argspec_tl
5029 {
5030   \int_if_zero:nTF \c@iRow
5031   { \@@_error:nn { in~first~row } \Vdots }
5032   {
5033     \int_compare:nNnTF \c@iRow = \l_@@_last_row_int
5034     { \@@_error:nn { in~last~row } \Vdots }
5035     {
5036       \@@_instruction_of_type:nnn \c_false_bool { Vdots }
5037       { #1 , down = #2 , up = #3 , middle = #4 }
5038     }
5039   }
5040   \bool_if:NF \l_@@_nullify_dots_bool
5041   { \phantom { \ensuremath { \@@_old_vdots } } }
5042   \bool_gset_true:N \g_@@_empty_cell_bool
5043 }

5044 \cs_new_protected:Npn \@@_Ddots
5045 { \@@_collect_options:n { \@@_Ddots_i } }
5046 \exp_args:NNo \NewDocumentCommand \@@_Ddots_i \l_@@_argspec_tl
5047 {

```

```

5048 \int_case:nnF \c@iRow
5049 {
5050     0 { \@@_error:nn { in~first~row } \Ddots }
5051     \l_@@_last_row_int { \@@_error:nn { in~last~row } \Ddots }
5052 }
5053 {
5054     \int_case:nnF \c@jCol
5055     {
5056         0 { \@@_error:nn { in~first~col } \Ddots }
5057         \l_@@_last_col_int { \@@_error:nn { in~last~col } \Ddots }
5058     }
5059     {
5060         \keys_set_known:nn { nicematrix / Ddots } { #1 }
5061         \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Ddots }
5062         { #1 , down = #2 , up = #3 , middle = #4 }
5063     }
5064 }
5065 \bool_if:NF \l_@@_nullify_dots_bool
5066 { \phantom { \ensuremath { \@@_old_ddots } } }
5067 \bool_gset_true:N \g_@@_empty_cell_bool
5068 }
5069
5070 \cs_new_protected:Npn \@@_Iddots
5071 { \@@_collect_options:n { \@@_Iddots_i } }
5072 \exp_args:NNo \NewDocumentCommand \@@_Iddots_i \l_@@_argspec_tl
5073 {
5074     \int_case:nnF \c@iRow
5075     {
5076         0 { \@@_error:nn { in~first~row } \Iddots }
5077         \l_@@_last_row_int { \@@_error:nn { in~last~row } \Iddots }
5078     }
5079     {
5080         \int_case:nnF \c@jCol
5081         {
5082             0 { \@@_error:nn { in~first~col } \Iddots }
5083             \l_@@_last_col_int { \@@_error:nn { in~last~col } \Iddots }
5084         }
5085         {
5086             \keys_set_known:nn { nicematrix / Ddots } { #1 }
5087             \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Iddots }
5088             { #1 , down = #2 , up = #3 , middle = #4 }
5089         }
5090     }
5091     \bool_if:NF \l_@@_nullify_dots_bool
5092     { \phantom { \ensuremath { \@@_old_iddots } } }
5093     \bool_gset_true:N \g_@@_empty_cell_bool
5094 }
5095 }

```

End of the \AddToHook.

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

```

5096 \keys_define:nn { nicematrix / Ddots }
5097 {
5098     draw-first .bool_set:N = \l_@@_draw_first_bool ,
5099     draw-first .default:n = true ,
5100     draw-first .value_forbidden:n = true
5101 }

```

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

```

5102 \cs_new_protected:Npn \@@_Hspace:

```

```

5103 {
5104   \bool_gset_true:N \g_@@_empty_cell_bool
5105   \hspace
5106 }

```

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`.

```

5107 \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`.

```

5108 \cs_new:Npn \@@_Hdotsfor:
5109 {
5110   \bool_lazy_and:nnTF
5111     { \int_if_zero_p:n \c@jCol }
5112     { \int_if_zero_p:n \l_@@_first_col_int }
5113     {
5114       \bool_if:NTF \g_@@_after_col_zero_bool
5115       {
5116         \multicolumn { 1 } { c } { }
5117         \@@_Hdotsfor_i
5118       }
5119       { \@@_fatal:n { Hdotsfor~in~col~0 } }
5120     }
5121   {
5122     \multicolumn { 1 } { c } { }
5123     \@@_Hdotsfor_i
5124   }
5125 }

```

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:`).

```

5126 \hook_gput_code:nnn { begindocument } { . }
5127 {
5128   \cs_set_nopar:Npn \l_@@_argspec_tl { m m 0 { } E { _ ^ : } { { } { } { } } }
5129   \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.

```

5130   \cs_new_protected:Npn \@@_Hdotsfor_i
5131     { \@@_collect_options:n { \@@_Hdotsfor_ii } }
5132   \exp_args:NNo \NewDocumentCommand \@@_Hdotsfor_ii \l_@@_argspec_tl
5133     {
5134       \tl_gput_right:Ne \g_@@_HVDotsfor_lines_tl
5135       {
5136         \@@_Hdotsfor:nnnn
5137         { \int_use:N \c@iRow }
5138         { \int_use:N \c@jCol }
5139         { #2 }
5140         {
5141           #1 , #3 ,
5142           down = \exp_not:n { #4 } ,
5143           up = \exp_not:n { #5 } ,
5144           middle = \exp_not:n { #6 }
5145         }
5146       }
5147       \prg_replicate:nn { #2 - 1 }
5148       {
5149         &
5150         \multicolumn { 1 } { c } { }

```

```

5151         \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
5152     }
5153 }
5154 }

```

```

5155 \cs_new_protected:Npn \@@_Hdotsfor:nmmm #1 #2 #3 #4
5156 {
5157     \bool_set_false:N \l_@@_initial_open_bool
5158     \bool_set_false:N \l_@@_final_open_bool

```

For the row, it's easy.

```

5159     \int_set:Nn \l_@@_initial_i_int { #1 }
5160     \int_set_eq:NN \l_@@_final_i_int \l_@@_initial_i_int

```

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

```

5161     \int_compare:nNnTF { #2 } = \c_one_int
5162     {
5163         \int_set_eq:NN \l_@@_initial_j_int \c_one_int
5164         \bool_set_true:N \l_@@_initial_open_bool
5165     }
5166     {
5167         \cs_if_exist:cTF
5168         {
5169             pgf @ sh @ ns @ \@@_env:
5170             - \int_use:N \l_@@_initial_i_int
5171             - \int_eval:n { #2 - 1 }
5172         }
5173         { \int_set:Nn \l_@@_initial_j_int { #2 - 1 } }
5174         {
5175             \int_set:Nn \l_@@_initial_j_int { #2 }
5176             \bool_set_true:N \l_@@_initial_open_bool
5177         }
5178     }
5179     \int_compare:nNnTF { #2 + #3 - 1 } = \c_jCol
5180     {
5181         \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5182         \bool_set_true:N \l_@@_final_open_bool
5183     }
5184     {
5185         \cs_if_exist:cTF
5186         {
5187             pgf @ sh @ ns @ \@@_env:
5188             - \int_use:N \l_@@_final_i_int
5189             - \int_eval:n { #2 + #3 }
5190         }
5191         { \int_set:Nn \l_@@_final_j_int { #2 + #3 } }
5192         {
5193             \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5194             \bool_set_true:N \l_@@_final_open_bool
5195         }
5196     }
5197 \group_begin:
5198 \@@_open_shorten:
5199 \int_if_zero:nTF { #1 }
5200 { \color { nicematrix-first-row } }
5201 {
5202     \int_compare:nNnT { #1 } = \g_@@_row_total_int
5203     { \color { nicematrix-last-row } }
5204 }
5205
5206 \keys_set:nn { nicematrix / xdots } { #4 }
5207 \tl_if_empty:oF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
5208 \@@_actually_draw_ldots:
5209 \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).

```

5210 \int_step_inline:nnn { #2 } { #2 + #3 - 1 }
5211 { \cs_set:cpn { @@ _ dotted _ #1 - ##1 } { } }
5212 }

5213 \hook_gput_code:nnn { begindocument } { . }
5214 {
5215   \cs_set_nopar:Npn \l_@@_argspec_tl { m m 0 { } E { _ ^ : } { { } { } { } } }
5216   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
5217   \cs_new_protected:Npn \@@_Vdotsfor:
5218     { \@@_collect_options:n { \@@_Vdotsfor_i } }
5219   \exp_args:NNo \NewDocumentCommand \@@_Vdotsfor_i \l_@@_argspec_tl
5220     {
5221       \bool_gset_true:N \g_@@_empty_cell_bool
5222       \tl_gput_right:Ne \g_@@_HVDotsfor_lines_tl
5223         {
5224           \@@_Vdotsfor:nnnn
5225             { \int_use:N \c@iRow }
5226             { \int_use:N \c@jCol }
5227             { #2 }
5228             {
5229               #1 , #3 ,
5230               down = \exp_not:n { #4 } ,
5231               up = \exp_not:n { #5 } ,
5232               middle = \exp_not:n { #6 }
5233             }
5234           }
5235         }
5236     }

```

```

5237 \cs_new_protected:Npn \@@_Vdotsfor:nnnn #1 #2 #3 #4
5238 {
5239   \bool_set_false:N \l_@@_initial_open_bool
5240   \bool_set_false:N \l_@@_final_open_bool

```

For the column, it’s easy.

```

5241 \int_set:Nn \l_@@_initial_j_int { #2 }
5242 \int_set_eq:NN \l_@@_final_j_int \l_@@_initial_j_int

```

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

```

5243 \int_compare:nNnTF { #1 } = \c_one_int
5244 {
5245   \int_set_eq:NN \l_@@_initial_i_int \c_one_int
5246   \bool_set_true:N \l_@@_initial_open_bool
5247 }
5248 {
5249   \cs_if_exist:cTF
5250     {
5251       pgf @ sh @ ns @ \@@_env:
5252       - \int_eval:n { #1 - 1 }
5253       - \int_use:N \l_@@_initial_j_int
5254     }
5255     { \int_set:Nn \l_@@_initial_i_int { #1 - 1 } }
5256     {
5257       \int_set:Nn \l_@@_initial_i_int { #1 }
5258       \bool_set_true:N \l_@@_initial_open_bool
5259     }
5260 }
5261 \int_compare:nNnTF { #1 + #3 - 1 } = \c@iRow
5262 {

```

```

5263 \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5264 \bool_set_true:N \l_@@_final_open_bool
5265 }
5266 {
5267 \cs_if_exist:cTF
5268 {
5269 pgf @ sh @ ns @ \@@_env:
5270 - \int_eval:n { #1 + #3 }
5271 - \int_use:N \l_@@_final_j_int
5272 }
5273 { \int_set:Nn \l_@@_final_i_int { #1 + #3 } }
5274 {
5275 \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5276 \bool_set_true:N \l_@@_final_open_bool
5277 }
5278 }
5279 \group_begin:
5280 \@@_open_shorten:
5281 \int_if_zero:nTF { #2 }
5282 { \color { nicematrix-first-col } }
5283 {
5284 \int_compare:nNnT { #2 } = \g_@@_col_total_int
5285 { \color { nicematrix-last-col } }
5286 }
5287 \keys_set:nn { nicematrix / xdots } { #4 }
5288 \tl_if_empty:oF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
5289 \@@_actually_draw_Vdots:
5290 \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).

```

5291 \int_step_inline:nnn { #1 } { #1 + #3 - 1 }
5292 { \cs_set:cpn { @@ _ dotted _ ##1 - #2 } { } }
5293 }

```

The command `\@@_rotate:` will be linked to `\rotate` in `{NiceArrayWithDelims}`.

```

5294 \NewDocumentCommand \@@_rotate: { 0 { } }
5295 {
5296 \peek_remove_spaces:n
5297 {
5298 \bool_gset_true:N \g_@@_rotate_bool
5299 \keys_set:nn { nicematrix / rotate } { #1 }
5300 }
5301 }
5302 \keys_define:nn { nicematrix / rotate }
5303 {
5304 c .code:n = \bool_gset_true:N \g_@@_rotate_c_bool ,
5305 c .value_forbidden:n = true ,
5306 unknown .code:n = \@@_error:n { Unknown~key~for~rotate }
5307 }

```

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

```
5308 \cs_new:Npn \@@_double_int_eval:n #1-#2 \q_stop
5309 {
5310   \tl_if_empty:nTF { #2 }
5311     { #1 }
5312     { \@@_double_int_eval_i:n #1-#2 \q_stop }
5313 }
5314 \cs_new:Npn \@@_double_int_eval_i:n #1-#2- \q_stop
5315 { \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).

```
5316 \hook_gput_code:nnn { begindocument } { . }
5317 {
5318   \cs_set_nopar:Npn \l_@@_argspec_tl
5319     { 0 { } m m ! 0 { } E { _ ^ : } { { } { } { } { } } }
5320   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
5321   \exp_args:NNo \NewDocumentCommand \@@_line \l_@@_argspec_tl
5322     {
5323       \group_begin:
5324       \keys_set:nn { nicematrix / xdots } { #1 , #4 , down = #5 , up = #6 }
5325       \tl_if_empty:oF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
5326       \use:e
5327       {
5328         \@@_line_i:nn
5329         { \@@_double_int_eval:n #2 - \q_stop }
5330         { \@@_double_int_eval:n #3 - \q_stop }
5331       }
5332       \group_end:
5333     }
5334 }
5335 \cs_new_protected:Npn \@@_line_i:nn #1 #2
5336 {
5337   \bool_set_false:N \l_@@_initial_open_bool
5338   \bool_set_false:N \l_@@_final_open_bool
5339   \bool_lazy_or:nnTF
5340     { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #1 } }
5341     { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #2 } }
5342     { \@@_error:nnn { unknown-cell-for-line-in~CodeAfter } { #1 } { #2 } }
5343   { \legacy_if:nF { measuring@ } { \@@_draw_line_ii:nn { #1 } { #2 } } }
5344 }
```

The test of `measuring@` is a security (cf. question 686649 on TeX StackExchange).

¹³Indeed, we want that the user may use the command `\line` in `\CodeAfter` with LaTeX counters in the arguments — with the command `\value`.

```

5345 \hook_gput_code:nnn { begindocument } { . }
5346 {
5347   \cs_new_protected:Npe \@@_draw_line_ii:nn #1 #2
5348   {

```

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:`.

```

5349     \c_@@_pgfortikzpicture_tl
5350     \@@_draw_line_iii:nn { #1 } { #2 }
5351     \c_@@_endpgfortikzpicture_tl
5352   }
5353 }

```

The following command *must* be protected (it’s used in the construction of `\@@_draw_line_ii:nn`).

```

5354 \cs_new_protected:Npn \@@_draw_line_iii:nn #1 #2
5355 {
5356   \pgfrememberpicturepositiononpagetrue
5357   \pgfpointshapeborder { \@@_env: - #1 } { \@@_qpoint:n { #2 } }
5358   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
5359   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
5360   \pgfpointshapeborder { \@@_env: - #2 } { \@@_qpoint:n { #1 } }
5361   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
5362   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
5363   \@@_draw_line:
5364 }

```

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`

```

5365 \cs_new:Npn \@@_if_row_less_than:nn #1 #2
5366 { \int_compare:nNnT { \c@iRow } < { #1 } { #2 } }

```

`\@@_put_in_row_style` will be used several times by `\RowStyle`.

```

5367 \cs_generate_variant:Nn \@@_put_in_row_style:n { e }
5368 \cs_set_protected:Npn \@@_put_in_row_style:n #1
5369 {
5370   \tl_gput_right:Ne \g_@@_row_style_tl
5371   {

```

Be careful, `\exp_not:N \@@_if_row_less_than:nn` can’t be replaced by a protected version of `\@@_if_row_less_than:nn`.

```

5372     \exp_not:N
5373     \@@_if_row_less_than:nn
5374     { \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`).

```

5375         { \exp_not:n { #1 } \scan_stop: }
5376     }
5377 }

5378 \keys_define:nn { nicematrix / RowStyle }
5379 {
5380     cell-space-top-limit .dim_set:N = \l_tmpa_dim ,
5381     cell-space-top-limit .value_required:n = true ,
5382     cell-space-bottom-limit .dim_set:N = \l_tmpb_dim ,
5383     cell-space-bottom-limit .value_required:n = true ,
5384     cell-space-limits .meta:n =
5385     {
5386         cell-space-top-limit = #1 ,
5387         cell-space-bottom-limit = #1 ,
5388     } ,
5389     color .tl_set:N = \l_@@_color_tl ,
5390     color .value_required:n = true ,
5391     bold .bool_set:N = \l_@@_bold_row_style_bool ,
5392     bold .default:n = true ,
5393     nb-rows .code:n =
5394         \str_if_eq:nnTF { #1 } { * }
5395         { \int_set:Nn \l_@@_key_nb_rows_int { 500 } }
5396         { \int_set:Nn \l_@@_key_nb_rows_int { #1 } } ,
5397     nb-rows .value_required:n = true ,
5398     rowcolor .tl_set:N = \l_tmpa_tl ,
5399     rowcolor .value_required:n = true ,
5400     unknown .code:n = \@@_error:n { Unknown~key~for~RowStyle }
5401 }

```

```

5402 \NewDocumentCommand \@@_RowStyle:n { 0 { } m }
5403 {
5404     \group_begin:
5405     \tl_clear:N \l_tmpa_tl
5406     \tl_clear:N \l_@@_color_tl
5407     \int_set_eq:NN \l_@@_key_nb_rows_int \c_one_int
5408     \dim_zero:N \l_tmpa_dim
5409     \dim_zero:N \l_tmpb_dim
5410     \keys_set:nn { nicematrix / RowStyle } { #1 }

```

If the key `rowcolor` has been used.

```

5411     \tl_if_empty:NF \l_tmpa_tl
5412     {

```

First, the end of the current row (we remind that `\RowStyle` applies to the *end* of the current row).

```

5413         \tl_gput_right:Ne \g_@@_pre_code_before_tl
5414         {

```

The command `\@@_exp_color_arg:No` is *fully expandable*.

```

5415             \@@_exp_color_arg:No \@@_rectanglecolor \l_tmpa_tl
5416             { \int_use:N \c@iRow - \int_use:N \c@jCol }
5417             { \int_use:N \c@iRow - * }
5418         }

```

Then, the other rows (if there is several rows).

```

5419         \int_compare:nNnT \l_@@_key_nb_rows_int > \c_one_int
5420         {
5421             \tl_gput_right:Ne \g_@@_pre_code_before_tl
5422             {
5423                 \@@_exp_color_arg:No \@@_rowcolor \l_tmpa_tl
5424                 {
5425                     \int_eval:n { \c@iRow + 1 }

```

```

5426         - \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int - 1 }
5427     }
5428 }
5429 }
5430 }
5431 \@@_put_in_row_style:n { \exp_not:n { #2 } }

```

\l_tmpa_dim is the value of the key cell-space-top-limit of \RowStyle.

```

5432 \dim_compare:nNnT \l_tmpa_dim > \c_zero_dim
5433 {
5434     \@@_put_in_row_style:e
5435     {
5436         \tl_gput_right:Nn \exp_not:N \g_@@_cell_after_hook_tl
5437         {

```

It's not possible to change the following code by using \dim_set_eq:NN (because of expansion).

```

5438         \dim_set:Nn \l_@@_cell_space_top_limit_dim
5439         { \dim_use:N \l_tmpa_dim }
5440     }
5441 }
5442 }

```

\l_tmpb_dim is the value of the key cell-space-bottom-limit of \RowStyle.

```

5443 \dim_compare:nNnT \l_tmpb_dim > \c_zero_dim
5444 {
5445     \@@_put_in_row_style:e
5446     {
5447         \tl_gput_right:Nn \exp_not:N \g_@@_cell_after_hook_tl
5448         {
5449             \dim_set:Nn \l_@@_cell_space_bottom_limit_dim
5450             { \dim_use:N \l_tmpb_dim }
5451         }
5452     }
5453 }

```

\l_@@_color_tl is the value of the key color of \RowStyle.

```

5454 \tl_if_empty:NF \l_@@_color_tl
5455 {
5456     \@@_put_in_row_style:e
5457     {
5458         \mode_leave_vertical:
5459         \@@_color:n { \l_@@_color_tl }
5460     }
5461 }

```

\l_@@_bold_row_style_bool is the value of the key bold.

```

5462 \bool_if:NT \l_@@_bold_row_style_bool
5463 {
5464     \@@_put_in_row_style:n
5465     {
5466         \exp_not:n
5467         {
5468             \if_mode_math:
5469                 \c_math_toggle_token
5470                 \bfseries \boldmath
5471                 \c_math_toggle_token
5472             \else:
5473                 \bfseries \boldmath
5474             \fi:
5475         }
5476     }
5477 }
5478 \group_end:
5479 \g_@@_row_style_tl
5480 \ignorespaces
5481 }

```

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).

```
5482 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e }
5483 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e e }
5484 \cs_new_protected:Npn \@@_add_to_colors_seq:nn #1 #2
5485 {
```

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.

```
5486 \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`.

```
5487 \str_if_in:nnF { #1 } { !! }
5488 {
5489 \seq_map_indexed_inline:Nn \g_@@_colors_seq
```

We use `\str_if_eq:nnTF` which is slightly faster than `\tl_if_eq:nnTF`.

```
5490 { \str_if_eq:nnT { #1 } { ##2 } { \int_set:Nn \l_tmpa_int { ##1 } } }
5491 }
5492 \int_if_zero:nTF \l_tmpa_int
```

First, the case where the color is a *new* color (not in the sequence).

```
5493 {
5494 \seq_gput_right:Nn \g_@@_colors_seq { #1 }
5495 \tl_gset:ce { g_@@_color _ \seq_count:N \g_@@_colors_seq _ tl } { #2 }
5496 }
```

Now, the case where the color is *not* a new color (the color is in the sequence at the position `\l_tmpa_int`).

```
5497 { \tl_gput_right:ce { g_@@_color _ \int_use:N \l_tmpa_int _ tl } { #2 } }
5498 }
```

The following command must be used within a `\pgfpicture`.

```
5499 \cs_new_protected:Npn \@@_clip_with_rounded_corners:
5500 {
5501 \dim_compare:nNnT \l_@@_tab_rounded_corners_dim > \c_zero_dim
5502 {
```

The TeX group is for `\pgfsetcornersarced` (whose scope is the TeX scope).

```

5503     \group_begin:
5504     \pgfsetcornersarced
5505     {
5506         \pgfpoint
5507         { \l_@@_tab_rounded_corners_dim }
5508         { \l_@@_tab_rounded_corners_dim }
5509     }

```

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.

```

5510     \bool_if:NTF \l_@@_hvlines_bool
5511     {
5512         \pgfpathrectanglecorners
5513         {
5514             \pgfpointadd
5515             { \@@_qpoint:n { row-1 } }
5516             { \pgfpoint { 0.5 \arrayrulewidth } { \c_zero_dim } }
5517         }
5518         {
5519             \pgfpointadd
5520             {
5521                 \@@_qpoint:n
5522                 { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
5523             }
5524             { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
5525         }
5526     }
5527     {
5528         \pgfpathrectanglecorners
5529         { \@@_qpoint:n { row-1 } }
5530         {
5531             \pgfpointadd
5532             {
5533                 \@@_qpoint:n
5534                 { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
5535             }
5536             { \pgfpoint \c_zero_dim \arrayrulewidth }
5537         }
5538     }
5539     \pgfusepath { clip }
5540     \group_end:

```

The TeX group was for `\pgfsetcornersarced`.

```

5541     }
5542 }

```

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`).

```

5543 \cs_new_protected:Npn \@@_actually_color:
5544 {
5545     \pgfpicture
5546     \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.

```

5547     \@@_clip_with_rounded_corners:
5548     \seq_map_indexed_inline:Nn \g_@@_colors_seq
5549     {
5550         \int_compare:nNnTF { ##1 } = \c_one_int

```

```

5551     {
5552         \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_nocolor:n
5553         \use:c { g_@@_color _ 1 _tl }
5554         \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_normal:n
5555     }
5556     {
5557         \begin { pgfscope }
5558             \@@_color_opacity ##2
5559             \use:c { g_@@_color _ ##1 _tl }
5560             \tl_gclear:c { g_@@_color _ ##1 _tl }
5561             \pgfusepath { fill }
5562         \end { pgfscope }
5563     }
5564 }
5565 \endpgfpicture
5566 }

```

The following command will extract the potential key `opacity` in its optional argument (between square brackets) and (of course) then apply the command `\color`.

```

5567 \cs_new_protected:Npn \@@_color_opacity
5568 {
5569     \peek_meaning:NTF [
5570         { \@@_color_opacity:w }
5571         { \@@_color_opacity:w [ ] }
5572     }

```

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.

```

5573 \cs_new_protected:Npn \@@_color_opacity:w [ #1 ]
5574 {
5575     \tl_clear:N \l_tmpa_tl
5576     \keys_set_known:nnN { nicematrix / color-opacity } { #1 } \l_tmpb_tl

```

`\l_tmpa_tl` (if not empty) is now the opacity and `\l_tmpb_tl` (if not empty) is now the colorimetric space.

```

5577     \tl_if_empty:NF \l_tmpa_tl { \exp_args:No \pgfsetfillopacity \l_tmpa_tl }
5578     \tl_if_empty:NTF \l_tmpb_tl
5579         { \@declaredcolor }
5580         { \use:e { \exp_not:N \@undeclaredcolor [ \l_tmpb_tl ] } }
5581 }

```

The following set of keys is used by the command `\@@_color_opacity:wn`.

```

5582 \keys_define:nn { nicematrix / color-opacity }
5583 {
5584     opacity .tl_set:N          = \l_tmpa_tl ,
5585     opacity .value_required:n = true
5586 }

```

```

5587 \cs_new_protected:Npn \@@_cartesian_color:nn #1 #2
5588 {
5589     \cs_set_nopar:Npn \l_@@_rows_tl { #1 }
5590     \cs_set_nopar:Npn \l_@@_cols_tl { #2 }
5591     \@@_cartesian_path:
5592 }

```

Here is an example : `\@@_rowcolor {red!15} {1,3,5-7,10-}`

```

5593 \NewDocumentCommand \@@_rowcolor { 0 { } m m }
5594 {
5595     \tl_if_blank:nF { #2 }
5596     {

```

```

5597 \@@_add_to_colors_seq:en
5598 { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5599 { \@@_cartesian_color:nn { #3 } { - } }
5600 }
5601 }

```

Here an example : `\@@_columncolor:nn {red!15} {1,3,5-7,10-}`

```

5602 \NewDocumentCommand \@@_columncolor { 0 { } m m }
5603 {
5604   \tl_if_blank:nF { #2 }
5605   {
5606     \@@_add_to_colors_seq:en
5607     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5608     { \@@_cartesian_color:nn { - } { #3 } }
5609   }
5610 }

```

Here is an example : `\@@_rectanglecolor{red!15}{2-3}{5-6}`

```

5611 \NewDocumentCommand \@@_rectanglecolor { 0 { } m m m }
5612 {
5613   \tl_if_blank:nF { #2 }
5614   {
5615     \@@_add_to_colors_seq:en
5616     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5617     { \@@_rectanglecolor:nnn { #3 } { #4 } { \c_zero_dim } }
5618   }
5619 }

```

The last argument is the radius of the corners of the rectangle.

```

5620 \NewDocumentCommand \@@_roundedrectanglecolor { 0 { } m m m m }
5621 {
5622   \tl_if_blank:nF { #2 }
5623   {
5624     \@@_add_to_colors_seq:en
5625     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5626     { \@@_rectanglecolor:nnn { #3 } { #4 } { #5 } }
5627   }
5628 }

```

The last argument is the radius of the corners of the rectangle.

```

5629 \cs_new_protected:Npn \@@_rectanglecolor:nnn #1 #2 #3
5630 {
5631   \@@_cut_on_hyphen:w #1 \q_stop
5632   \tl_clear_new:N \l_@@_tmpc_tl
5633   \tl_clear_new:N \l_@@_tmpd_tl
5634   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
5635   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
5636   \@@_cut_on_hyphen:w #2 \q_stop
5637   \tl_set:Ne \l_@@_rows_tl { \l_@@_tmpc_tl - \l_tmpa_tl }
5638   \tl_set:Ne \l_@@_cols_tl { \l_@@_tmpd_tl - \l_tmpb_tl }

```

The command `\@@_cartesian_path:n` takes in two implicit arguments: `\l_@@_cols_tl` and `\l_@@_rows_tl`.

```

5639 \@@_cartesian_path:n { #3 }
5640 }

```

Here is an example : `\@@_cellcolor[rgb]{0.5,0.5,0}{2-3,3-4,4-5,5-6}`

```

5641 \NewDocumentCommand \@@_cellcolor { 0 { } m m }
5642 {
5643   \clist_map_inline:nn { #3 }
5644   { \@@_rectanglecolor [ #1 ] { #2 } { ##1 } { ##1 } }
5645 }

```

```

5646 \NewDocumentCommand \@@_chessboardcolors { 0 { } m m }
5647 {
5648   \int_step_inline:nn \c@iRow
5649   {
5650     \int_step_inline:nn \c@jCol
5651     {
5652       \int_if_even:nTF { #####1 + ##1 }
5653       { \@@_cellcolor [ #1 ] { #2 } }
5654       { \@@_cellcolor [ #1 ] { #3 } }
5655       { ##1 - #####1 }
5656     }
5657   }
5658 }

```

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”.

```

5659 \NewDocumentCommand \@@_arraycolor { 0 { } m }
5660 {
5661   \@@_rectanglecolor [ #1 ] { #2 }
5662   { 1 - 1 }
5663   { \int_use:N \c@iRow - \int_use:N \c@jCol }
5664 }

```

```

5665 \keys_define:nn { nicematrix / rowcolors }
5666 {
5667   respect-blocks .bool_set:N = \l_@@_respect_blocks_bool ,
5668   respect-blocks .default:n = true ,
5669   cols .tl_set:N = \l_@@_cols_tl ,
5670   restart .bool_set:N = \l_@@_rowcolors_restart_bool ,
5671   restart .default:n = true ,
5672   unknown .code:n = \@@_error:n { Unknown~key~for~rowcolors }
5673 }

```

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*.

```

5674 \NewDocumentCommand \@@_rowlistcolors { 0 { } m m 0 { } }
5675 {

```

The group is for the options. `\l_@@_colors_seq` will be the list of colors.

```

5676   \group_begin:
5677   \seq_clear_new:N \l_@@_colors_seq
5678   \seq_set_split:Nnn \l_@@_colors_seq { , } { #3 }
5679   \tl_clear_new:N \l_@@_cols_tl
5680   \cs_set_nopar:Npn \l_@@_cols_tl { - }
5681   \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).

```

5682   \int_zero_new:N \l_@@_color_int
5683   \int_set_eq:NN \l_@@_color_int \c_one_int
5684   \bool_if:NT \l_@@_respect_blocks_bool
5685   {

```

We don't want to take into account a block which is completely in the “first column” (number 0) or in the “last column” and that's why we filter the sequence of the blocks (in the sequence `\l_tmpa_seq`).

```

5686     \seq_set_eq:NN \l_tmpb_seq \g_@@_pos_of_blocks_seq
5687     \seq_set_filter:NNn \l_tmpa_seq \l_tmpb_seq
5688     { \@@_not_in_exterior_p:nnnnn ##1 }
5689   }
5690   \pgfpicture
5691   \pgf@relevantforpicturesizefalse

```

#2 is the list of intervals of rows.

```

5692   \clist_map_inline:nn { #2 }
5693   {
5694     \cs_set_nopar:Npn \l_tmpa_tl { ##1 }
5695     \tl_if_in:NnTF \l_tmpa_tl { - }
5696     { \@@_cut_on_hyphen:w ##1 \q_stop }
5697     { \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.

```

5698     \int_set:Nn \l_tmpa_int \l_tmpa_tl
5699     \int_set:Nn \l_@@_color_int
5700     { \bool_if:NTF \l_@@_rowcolors_restart_bool 1 \l_tmpa_tl }
5701     \int_zero_new:N \l_@@_tmpc_int
5702     \int_set:Nn \l_@@_tmpc_int \l_tmpb_tl
5703     \int_do_until:nNnn \l_tmpa_int > \l_@@_tmpc_int
5704     {

```

We will compute in `\l_tmpb_int` the last row of the “block”.

```

5705     \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).

```

5706     \bool_if:NT \l_@@_respect_blocks_bool
5707     {
5708       \seq_set_filter:NNn \l_tmpb_seq \l_tmpa_seq
5709       { \@@_intersect_our_row_p:nnnnn #####1 }
5710       \seq_map_inline:Nn \l_tmpb_seq { \@@_rowcolors_i:nnnnn #####1 }

```

Now, the last row of the block is computed in `\l_tmpb_int`.

```

5711     }
5712     \tl_set:No \l_@@_rows_tl
5713     { \int_use:N \l_tmpa_int - \int_use:N \l_tmpb_int }

```

`\l_@@_tmpc_tl` will be the color that we will use.

```

5714     \tl_clear_new:N \l_@@_color_tl
5715     \tl_set:Ne \l_@@_color_tl
5716     {
5717       \@@_color_index:n
5718       {
5719         \int_mod:nn
5720         { \l_@@_color_int - 1 }
5721         { \seq_count:N \l_@@_colors_seq }
5722         + 1
5723       }
5724     }
5725     \tl_if_empty:NF \l_@@_color_tl
5726     {
5727       \@@_add_to_colors_seq:ee
5728       { \tl_if_blank:nF { #1 } { [ #1 ] } { \l_@@_color_tl } }
5729       { \@@_cartesian_color:nn { \l_@@_rows_tl } { \l_@@_cols_tl } }
5730     }
5731     \int_incr:N \l_@@_color_int
5732     \int_set:Nn \l_tmpa_int { \l_tmpb_int + 1 }
5733   }
5734 }
5735 \endpgfpicture

```



```

5736 \group_end:
5737 }

```

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.

```

5738 \cs_new:Npn \@@_color_index:n #1
5739 {
5740   \str_if_eq:eeTF { \seq_item:Nn \l_@@_colors_seq { #1 } } { = }
5741   { \@@_color_index:n { #1 - 1 } }
5742   { \seq_item:Nn \l_@@_colors_seq { #1 } }
5743 }

```

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.

```

5744 \NewDocumentCommand \@@_rowcolors { 0 { } m m m }
5745 { \@@_rowlistcolors [ #1 ] { #2 } { { #3 } , { #4 } } }

```

The braces around #3 and #4 are mandatory.

```

5746 \cs_new_protected:Npn \@@_rowcolors_i:nnnnn #1 #2 #3 #4 #5
5747 {
5748   \int_compare:nNtT { #3 } > \l_tmpb_int
5749   { \int_set:Nn \l_tmpb_int { #3 } }
5750 }

```

```

5751 \prg_new_conditional:Nnn \@@_not_in_exterior:nnnnn p
5752 {
5753   \int_if_zero:nTF { #4 }
5754   \prg_return_false:
5755   {
5756     \int_compare:nNtTF { #2 } > \c@jCol
5757     \prg_return_false:
5758     \prg_return_true:
5759   }
5760 }

```

The following command return true when the block intersects the row `\l_tmpa_int`.

```

5761 \prg_new_conditional:Nnn \@@_intersect_our_row:nnnnn p
5762 {
5763   \int_compare:nNtTF { #1 } > \l_tmpa_int
5764   \prg_return_false:
5765   {
5766     \int_compare:nNtTF \l_tmpa_int > { #3 }
5767     \prg_return_false:
5768     \prg_return_true:
5769   }
5770 }

```

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`).

```

5771 \cs_new_protected:Npn \@@_cartesian_path_normal:n #1
5772 {
5773   \dim_compare:nNtTF { #1 } = \c_zero_dim
5774   {
5775     \bool_if:NTF

```

```

5776 \l_@@_nocolor_used_bool
5777 \@@_cartesian_path_normal_ii:
5778 {
5779 \seq_if_empty:NTF \l_@@_corners_cells_seq
5780 { \@@_cartesian_path_normal_i:n { #1 } }
5781 \@@_cartesian_path_normal_ii:
5782 }
5783 }
5784 { \@@_cartesian_path_normal_i:n { #1 } }
5785 }

```

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.

```

5786 \cs_new_protected:Npn \@@_cartesian_path_normal_i:n #1
5787 {
5788 \pgfsetcornersarced { \pgfpoint { #1 } { #1 } }

```

We begin the loop over the columns.

```

5789 \clist_map_inline:Nn \l_@@_cols_tl
5790 {
5791 \cs_set_nopar:Npn \l_tmpa_tl { ##1 }
5792 \tl_if_in:NnTF \l_tmpa_tl { - }
5793 { \@@_cut_on_hyphen:w ##1 \q_stop }
5794 { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
5795 \tl_if_empty:NTF \l_tmpa_tl
5796 { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5797 {
5798 \tl_if_eq:NNT \l_tmpa_tl \c_@@_star_tl
5799 { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5800 }
5801 \tl_if_empty:NTF \l_tmpb_tl
5802 { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
5803 {
5804 \tl_if_eq:NNT \l_tmpb_tl \c_@@_star_tl
5805 { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
5806 }
5807 \int_compare:nNnT \l_tmpb_tl > \g_@@_col_total_int
5808 { \tl_set:No \l_tmpb_tl { \int_use:N \g_@@_col_total_int } }

```

\l_@@_tmpc_tl will contain the number of column.

```

5809 \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
5810 \@@_qpoint:n { col - \l_tmpa_tl }
5811 \int_compare:nNnTF \l_@@_first_col_int = \l_tmpa_tl
5812 { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
5813 { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
5814 \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
5815 \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }

```

We begin the loop over the rows.

```

5816 \clist_map_inline:Nn \l_@@_rows_tl
5817 {
5818 \cs_set_nopar:Npn \l_tmpa_tl { #####1 }
5819 \tl_if_in:NnTF \l_tmpa_tl { - }
5820 { \@@_cut_on_hyphen:w #####1 \q_stop }
5821 { \@@_cut_on_hyphen:w #####1 - #####1 \q_stop }
5822 \tl_if_empty:NTF \l_tmpa_tl
5823 { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5824 {
5825 \tl_if_eq:NNT \l_tmpa_tl \c_@@_star_tl
5826 { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5827 }
5828 \tl_if_empty:NTF \l_tmpb_tl
5829 { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }
5830 {

```

```

5831         \tl_if_eq:NNT \l_tmpb_tl \c_@@_star_tl
5832         { \tl_set:No \l_tmpb_tl { \int_use:N \c_iRow } }
5833     }
5834     \int_compare:nNt \l_tmpb_tl > \g_@@_row_total_int
5835     { \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`.

```

5836     \cs_if_exist:cF
5837     { @@ _ \l_tmpa_tl _ \l_@@_tmpc_tl _ nocolor }
5838     {
5839         \@@_qpoint:n { row - \int_eval:n { \l_tmpb_tl + 1 } }
5840         \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
5841         \@@_qpoint:n { row - \l_tmpa_tl }
5842         \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
5843         \pgfpathrectanglecorners
5844         { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
5845         { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5846     }
5847 }
5848 }
5849 }

```

Now, the case where the cells will be colored cell by cell (it's mandatory for example if the key `corners` is used).

```

5850 \cs_new_protected:Npn \@@_cartesian_path_normal_ii:
5851 {
5852     \@@_expand_clist:NN \l_@@_cols_tl \c_jCol
5853     \@@_expand_clist:NN \l_@@_rows_tl \c_iRow

```

We begin the loop over the columns.

```

5854     \clist_map_inline:Nn \l_@@_cols_tl
5855     {
5856         \@@_qpoint:n { col - ##1 }
5857         \int_compare:nNtF \l_@@_first_col_int = { ##1 }
5858         { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
5859         { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
5860         \@@_qpoint:n { col - \int_eval:n { ##1 + 1 } }
5861         \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }

```

We begin the loop over the rows.

```

5862     \clist_map_inline:Nn \l_@@_rows_tl
5863     {
5864         \seq_if_in:NnF \l_@@_corners_cells_seq
5865         { #####1 - ##1 }
5866         {
5867             \@@_qpoint:n { row - \int_eval:n { #####1 + 1 } }
5868             \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
5869             \@@_qpoint:n { row - #####1 }
5870             \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
5871             \cs_if_exist:cF { @@ _ #####1 _ ##1 _ nocolor }
5872             {
5873                 \pgfpathrectanglecorners
5874                 { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
5875                 { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5876             }
5877         }
5878     }
5879 }
5880 }

```

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`).

```

5881 \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

```

5882 \cs_new_protected:Npn \@@_cartesian_path_nocolor:n #1
5883 {
5884   \bool_set_true:N \l_@@_nocolor_used_bool
5885   \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
5886   \@@_expand_clist:NN \l_@@_rows_tl \c@iRow

```

We begin the loop over the columns.

```

5887   \clist_map_inline:Nn \l_@@_rows_tl
5888   {
5889     \clist_map_inline:Nn \l_@@_cols_tl
5890     { \cs_set:cpn { @@ _ ##1 _ #####1 _ nocolor } { } }
5891   }
5892 }

```

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`.

```

5893 \cs_new_protected:Npn \@@_expand_clist:NN #1 #2
5894 {
5895   \clist_set_eq:NN \l_tmpa_clist #1
5896   \clist_clear:N #1
5897   \clist_map_inline:Nn \l_tmpa_clist
5898   {
5899     \cs_set_nopar:Npn \l_tmpa_tl { ##1 }
5900     \tl_if_in:NnTF \l_tmpa_tl { - }
5901     { \@@_cut_on_hyphen:w ##1 \q_stop }
5902     { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
5903     \bool_lazy_or:nnT
5904     { \tl_if_blank_p:o \l_tmpa_tl }
5905     { \str_if_eq_p:on \l_tmpa_tl { * } }
5906     { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5907     \bool_lazy_or:nnT
5908     { \tl_if_blank_p:o \l_tmpb_tl }
5909     { \str_if_eq_p:on \l_tmpb_tl { * } }
5910     { \tl_set:No \l_tmpb_tl { \int_use:N #2 } }
5911     \int_compare:nNnT \l_tmpb_tl > #2
5912     { \tl_set:No \l_tmpb_tl { \int_use:N #2 } }
5913     \int_step_inline:nnn \l_tmpa_tl \l_tmpb_tl
5914     { \clist_put_right:Nn #1 { #####1 } }
5915   }
5916 }

```

When the user uses the key `color-inside`, the following command will be linked to `\cellcolor` in the tabular.

```

5917 \NewDocumentCommand \@@_cellcolor_tabular { 0 { } m }
5918 {
5919   \@@_test_color_inside:
5920   \tl_gput_right:Ne \g_@@_pre_code_before_tl
5921   {

```

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`).

```

5922     \@@_cellcolor [ #1 ] { \exp_not:n { #2 } }
5923     { \int_use:N \c@iRow - \int_use:N \c@jCol }
5924   }
5925   \ignorespaces
5926 }

```

When the user uses the key `color-inside`, the following command will be linked to `\rowcolor` in the tabular.

```

5927 \NewDocumentCommand \@@_rowcolor_tabular { 0 { } m }
5928 {
5929   \@@_test_color_inside:
5930   \tl_gput_right:Ne \g_@@_pre_code_before_tl
5931   {
5932     \@@_rectanglecolor [ #1 ] { \exp_not:n { #2 } }
5933     { \int_use:N \c@iRow - \int_use:N \c@jCol }
5934     { \int_use:N \c@iRow - \exp_not:n { \int_use:N \c@jCol } }
5935   }
5936   \ignorespaces
5937 }

```

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 currying).

```

5938 \NewDocumentCommand { \@@_rowcolors_tabular } { 0 { } m m }
5939 { \@@_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.

```

5940 \NewDocumentCommand { \@@_rowlistcolors_tabular } { 0 { } m 0 { } }
5941 {
5942   \@@_test_color_inside:
5943   \peek_remove_spaces:n
5944   { \@@_rowlistcolors_tabular:nnn { #1 } { #2 } { #3 } }
5945 }

```

```

5946 \cs_new_protected:Npn \@@_rowlistcolors_tabular:nnn #1 #2 #3
5947 {

```

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`.

```

5948   \seq_gclear:N \g_tmpa_seq
5949   \seq_map_inline:Nn \g_@@_rowlistcolors_seq
5950   { \@@_rowlistcolors_tabular_i:nnnn ##1 }
5951   \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`.

```

5952   \seq_gput_right:Ne \g_@@_rowlistcolors_seq
5953   {
5954     { \int_use:N \c@iRow }
5955     { \exp_not:n { #1 } }
5956     { \exp_not:n { #2 } }
5957     { restart , cols = \int_use:N \c@jCol - , \exp_not:n { #3 } }
5958   }
5959 }

```

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`).

```
5960 \cs_new_protected:Npn \@@_rowlistcolors_tabular_i:nnnn #1 #2 #3 #4
5961 {
5962   \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).

```
5963   { \seq_gput_right:Nn \g_tmpa_seq { { #1 } { #2 } { #3 } { #4 } } }
5964   {
5965     \tl_gput_right:Ne \g_@@_pre_code_before_tl
5966     {
5967       \@@_rowlistcolors
5968       [ \exp_not:n { #2 } ]
5969       { #1 - \int_eval:n { \c@iRow - 1 } }
5970       { \exp_not:n { #3 } }
5971       [ \exp_not:n { #4 } ]
5972     }
5973   }
5974 }
```

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.

```
5975 \cs_new_protected:Npn \@@_clear_rowlistcolors_seq:
5976 {
5977   \seq_map_inline:Nn \g_@@_rowlistcolors_seq
5978   { \@@_rowlistcolors_tabular_ii:nnnn ##1 }
5979   \seq_gclear:N \g_@@_rowlistcolors_seq
5980 }
```

```
5981 \cs_new_protected:Npn \@@_rowlistcolors_tabular_ii:nnnn #1 #2 #3 #4
5982 {
5983   \tl_gput_right:Nn \g_@@_pre_code_before_tl
5984   { \@@_rowlistcolors [ #2 ] { #1 } { #3 } [ #4 ] }
5985 }
```

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.

```
5986 \NewDocumentCommand \@@_columncolor_preamble { 0 { } m }
5987 {
```

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).

```
5988   \int_compare:nNnT \c@jCol > \g_@@_col_total_int
5989   {
```

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).

```
5990     \tl_gput_left:Ne \g_@@_pre_code_before_tl
5991     {
5992       \exp_not:N \columncolor [ #1 ]
5993       { \exp_not:n { #2 } } { \int_use:N \c@jCol }
5994     }
5995   }
5996 }
```

```

5997 \hook_gput_code:nnn { begindocument } { . }
5998 {
5999   \IfPackageLoadedTF { colortbl }
6000   {
6001     \cs_set_eq:NN \@@_old_cellcolor \cellcolor
6002     \cs_set_eq:NN \@@_old_rowcolor \rowcolor
6003     \cs_new_protected:Npn \@@_revert_colortbl:
6004     {
6005       \hook_gput_code:nnn { env / tabular / begin } { nicematrix }
6006       {
6007         \cs_set_eq:NN \cellcolor \@@_old_cellcolor
6008         \cs_set_eq:NN \rowcolor \@@_old_rowcolor
6009       }
6010     }
6011   }
6012   { \cs_new_protected:Npn \@@_revert_colortbl: { } }
6013 }

```

23 The vertical and horizontal rules

OnlyMainNiceMatrix

We give to the user the possibility to define new types of columns (with `\newcolumnstype` of `array`) for special vertical rules (*e.g.* rules thicker than the standard ones) which will not extend in the potential exterior rows of the array.

We provide the command `\OnlyMainNiceMatrix` in that goal. However, that command must be no-op outside the environments of `nicematrix` (and so the user will be allowed to use the same new type of column in the environments of `nicematrix` and in the standard environments of `array`).

That's why we provide first a global definition of `\OnlyMainNiceMatrix`.

```

6014 \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`.

```

6015 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix:n #1
6016 {
6017   \int_if_zero:nTF \l_@@_first_col_int
6018   { \@@_OnlyMainNiceMatrix_i:n { #1 } }
6019   {
6020     \int_if_zero:nTF \c@jCol
6021     {
6022       \int_compare:nNf \c@iRow = { -1 }
6023       { \int_compare:nNf \c@iRow = { \l_@@_last_row_int - 1 } { #1 } }
6024     }
6025     { \@@_OnlyMainNiceMatrix_i:n { #1 } }
6026   }
6027 }

```

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.

```

6028 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix_i:n #1
6029 {
6030   \int_if_zero:nF \c@iRow
6031   {
6032     \int_compare:nNf \c@iRow = \l_@@_last_row_int
6033     {

```

```

6034         \int_compare:nNtT \c@jCol > \c_zero_int
6035         { \bool_if:NF \l_@@_in_last_col_bool { #1 } }
6036     }
6037 }
6038 }

```

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:nNtT \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.

```

6039 \keys_define:nn { nicematrix / Rules }
6040 {
6041     position .int_set:N = \l_@@_position_int ,
6042     position .value_required:n = true ,
6043     start .int_set:N = \l_@@_start_int ,
6044     end .code:n =
6045         \bool_lazy_or:nnTF
6046         { \tl_if_empty_p:n { #1 } }
6047         { \str_if_eq_p:nn { #1 } { last } }
6048         { \int_set_eq:NN \l_@@_end_int \c@jCol }
6049         { \int_set:Nn \l_@@_end_int { #1 } }
6050 }

```

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.

```

6051 \keys_define:nn { nicematrix / RulesBis }
6052 {
6053     multiplicity .int_set:N = \l_@@_multiplicity_int ,
6054     multiplicity .initial:n = 1 ,
6055     dotted .bool_set:N = \l_@@_dotted_bool ,
6056     dotted .initial:n = false ,
6057     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]`.

```

6058     color .code:n =
6059         \@@_set_CT@arc@:n { #1 }
6060         \tl_set:Nn \l_@@_rule_color_tl { #1 } ,
6061     color .value_required:n = true ,
6062     sep-color .code:n = \@@_set_CT@drsc@:n { #1 } ,
6063     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.

```

6064     tikz .code:n =
6065         \IfPackageLoadedTF { tikz }
6066         { \clist_put_right:Nn \l_@@_tikz_rule_tl { #1 } }
6067         { \@@_error:n { tikz-without-tikz } } ,
6068     tikz .value_required:n = true ,
6069     total-width .dim_set:N = \l_@@_rule_width_dim ,

```



```

6070     total-width .value_required:n = true ,
6071     width .meta:n = { total-width = #1 } ,
6072     unknown .code:n = \@_error:n { Unknow~key~for~RulesBis }
6073 }

```

The vertical rules

The following command will be executed in the internal `\CodeAfter`. The argument `#1` is a list of `key=value` pairs.

```

6074 \cs_new_protected:Npn \@_vline:n #1
6075 {

```

The group is for the options.

```

6076     \group_begin:
6077     \int_set_eq:NN \l_@@_end_int \c@iRow
6078     \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).

```

6079     \int_compare:nNnT \l_@@_position_int < { \c@jCol + 2 }
6080     \@_vline_i:
6081     \group_end:
6082 }

```

```

6083 \cs_new_protected:Npn \@_vline_i:
6084 {

```

`\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`.

```

6085     \tl_set:Nn \l_tmpb_tl { \int_use:N \l_@@_position_int }
6086     \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
6087     \l_tmpa_tl
6088     {

```

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.

```

6089         \bool_gset_true:N \g_tmpa_bool
6090         \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6091         { \@_test_vline_in_block:nnnnn ##1 }
6092         \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
6093         { \@_test_vline_in_block:nnnnn ##1 }
6094         \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
6095         { \@_test_vline_in_stroken_block:nnnn ##1 }
6096         \clist_if_empty:NF \l_@@_corners_clist \@_test_in_corner_v:
6097         \bool_if:NTF \g_tmpa_bool
6098         {
6099             \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.

```

6100             { \int_set:Nn \l_@@_local_start_int \l_tmpa_tl }
6101         }
6102     {
6103         \int_compare:nNnT \l_@@_local_start_int > \c_zero_int
6104         {
6105             \int_set:Nn \l_@@_local_end_int { \l_tmpa_tl - 1 }
6106             \@_vline_ii:
6107             \int_zero:N \l_@@_local_start_int
6108         }
6109     }
6110 }
6111 \int_compare:nNnT \l_@@_local_start_int > \c_zero_int
6112 {

```

```

6113     \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
6114     \@@_vline_ii:
6115   }
6116 }

6117 \cs_new_protected:Npn \@@_test_in_corner_v:
6118 {
6119   \int_compare:nNnTF \l_tmpb_tl = { \int_eval:n { \c@jCol + 1 } }
6120   {
6121     \seq_if_in:NeT
6122       \l_@@_corners_cells_seq
6123       { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
6124       { \bool_set_false:N \g_tmpa_bool }
6125   }
6126   {
6127     \seq_if_in:NeT
6128       \l_@@_corners_cells_seq
6129       { \l_tmpa_tl - \l_tmpb_tl }
6130     {
6131       \int_compare:nNnTF \l_tmpb_tl = \c_one_int
6132       { \bool_set_false:N \g_tmpa_bool }
6133       {
6134         \seq_if_in:NeT
6135           \l_@@_corners_cells_seq
6136           { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
6137           { \bool_set_false:N \g_tmpa_bool }
6138       }
6139     }
6140   }
6141 }

6142 \cs_new_protected:Npn \@@_vline_ii:
6143 {
6144   \tl_clear:N \l_@@_tikz_rule_tl
6145   \keys_set:no { nicematrix / RulesBis } \l_@@_other_keys_tl
6146   \bool_if:NTF \l_@@_dotted_bool
6147     \@@_vline_iv:
6148     {
6149       \tl_if_empty:NTF \l_@@_tikz_rule_tl
6150       \@@_vline_iii:
6151       \@@_vline_v:
6152     }
6153 }

```

First the case of a standard rule: the user has not used the key dotted nor the key tikz.

```

6154 \cs_new_protected:Npn \@@_vline_iii:
6155 {
6156   \pgfpicture
6157   \pgfrememberpicturepositiononpagetrue
6158   \pgf@relevantforpicturesizefalse
6159   \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6160   \dim_set_eq:NN \l_tmpa_dim \pgf@y
6161   \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6162   \dim_set:Nn \l_tmpb_dim
6163   {
6164     \pgf@x
6165     - 0.5 \l_@@_rule_width_dim
6166     +
6167     ( \arrayrulewidth * \l_@@_multiplicity_int
6168       + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6169   }

```

```

6170 \l_@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6171 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
6172 \bool_lazy_all:nT
6173 {
6174   { \int_compare_p:nNn \l_@@_multiplicity_int > \c_one_int }
6175   { \cs_if_exist_p:N \CT@drsc@ }
6176   { ! \tl_if_blank_p:o \CT@drsc@ }
6177 }
6178 {
6179   \group_begin:
6180   \CT@drsc@
6181   \dim_add:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }
6182   \dim_sub:Nn \l_@@_tmpc_dim { 0.5 \arrayrulewidth }
6183   \dim_set:Nn \l_@@_tmpd_dim
6184   {
6185     \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
6186     * ( \l_@@_multiplicity_int - 1 )
6187   }
6188   \pgfpathrectanglecorners
6189   { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6190   { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
6191   \pgfusepath { fill }
6192   \group_end:
6193 }
6194 \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6195 \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
6196 \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6197 {
6198   \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
6199   \dim_sub:Nn \l_tmpb_dim \doublerulesep
6200   \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6201   \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
6202 }
6203 \CT@arc@
6204 \pgfsetlinewidth { 1.1 \arrayrulewidth }
6205 \pgfsetrectcap
6206 \pgfusepathqstroke
6207 \endpgfpicture
6208 }

```

The following code is for the case of a dotted rule (with our system of rounded dots).

```

6209 \cs_new_protected:Npn \l_@@_vline_iv:
6210 {
6211   \pgfpicture
6212   \pgfrememberpicturepositiononpagetrue
6213   \pgf@relevantforpicturesizefalse
6214   \l_@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6215   \dim_set:Nn \l_@@_x_initial_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
6216   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
6217   \l_@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6218   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
6219   \l_@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6220   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
6221   \CT@arc@
6222   \l_@@_draw_line:
6223   \endpgfpicture
6224 }

```

The following code is for the case when the user uses the key `tikz`.

```

6225 \cs_new_protected:Npn \l_@@_vline_v:
6226 {
6227   \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).

```

6228 \CT@arc@
6229 \tl_if_empty:NF \l_@@_rule_color_tl
6230 { \tl_put_right:Ne \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6231 \pgfrememberpicturepositiononpagetrue
6232 \pgf@relevantforpicturesizefalse
6233 \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6234 \dim_set_eq:NN \l_tmpa_dim \pgf@y
6235 \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6236 \dim_set:Nn \l_tmpb_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
6237 \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6238 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
6239 \exp_args:No \tikzset \l_@@_tikz_rule_tl
6240 \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6241 ( \l_tmpb_dim , \l_tmpa_dim ) --
6242 ( \l_tmpb_dim , \l_@@_tmpc_dim ) ;
6243 \end { tikzpicture }
6244 }

```

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).

```

6245 \cs_new_protected:Npn \@@_draw_vlines:
6246 {
6247   \int_step_inline:nnn
6248   { \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool 2 1 }
6249   {
6250     \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool
6251     \c@jCol
6252     { \int_eval:n { \c@jCol + 1 } }
6253   }
6254   {
6255     \tl_if_eq:NNF \l_@@_vlines_clist \c_@@_all_tl
6256     { \clist_if_in:NnT \l_@@_vlines_clist { ##1 } }
6257     { \@@_vline:n { position = ##1 , total-width = \arrayrulewidth } }
6258   }
6259 }

```

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}`.

```

6260 \cs_new_protected:Npn \@@_hline:n #1
6261 {
6262   The group is for the options.
6263   \group_begin:
6264   \int_zero_new:N \l_@@_end_int
6265   \int_set_eq:NN \l_@@_end_int \c@jCol
6266   \keys_set_known:nN { nicematrix / Rules } { #1 } \l_@@_other_keys_tl
6267   \@@_hline_i:
6268   \group_end:
6269 }
6270 \cs_new_protected:Npn \@@_hline_i:
6271 {
6272   \int_zero_new:N \l_@@_local_start_int
6273   \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.

```

6273 \tl_set:No \l_tmpa_tl { \int_use:N \l_@@_position_int }
6274 \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
6275 \l_tmpb_tl
6276 {

```

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.

```

6277 \bool_gset_true:N \g_tmpa_bool
6278 \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6279 { \@@_test_hline_in_block:nnnnn ##1 }
6280 \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
6281 { \@@_test_hline_in_block:nnnnn ##1 }
6282 \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
6283 { \@@_test_hline_in_stroken_block:nnnn ##1 }
6284 \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_h:
6285 \bool_if:NTF \g_tmpa_bool
6286 {
6287 \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.

```

6288 { \int_set:Nn \l_@@_local_start_int \l_tmpb_tl }
6289 }
6290 {
6291 \int_compare:nNnT \l_@@_local_start_int > \c_zero_int
6292 {
6293 \int_set:Nn \l_@@_local_end_int { \l_tmpb_tl - 1 }
6294 \@@_hline_ii:
6295 \int_zero:N \l_@@_local_start_int
6296 }
6297 }
6298 }
6299 \int_compare:nNnT \l_@@_local_start_int > \c_zero_int
6300 {
6301 \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
6302 \@@_hline_ii:
6303 }
6304 }

```

```

6305 \cs_new_protected:Npn \@@_test_in_corner_h:
6306 {
6307 \int_compare:nNnTF \l_tmpa_tl = { \int_eval:n { \c@iRow + 1 } }
6308 {
6309 \seq_if_in:NeT
6310 \l_@@_corners_cells_seq
6311 { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
6312 { \bool_set_false:N \g_tmpa_bool }
6313 }
6314 {
6315 \seq_if_in:NeT
6316 \l_@@_corners_cells_seq
6317 { \l_tmpa_tl - \l_tmpb_tl }
6318 {
6319 \int_compare:nNnTF \l_tmpa_tl = \c_one_int
6320 { \bool_set_false:N \g_tmpa_bool }
6321 {
6322 \seq_if_in:NeT
6323 \l_@@_corners_cells_seq
6324 { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }

```

```

6325         { \bool_set_false:N \g_tmpa_bool }
6326     }
6327 }
6328 }
6329 }

6330 \cs_new_protected:Npn \@@_hline_ii:
6331 {
6332     \tl_clear:N \l_@@_tikz_rule_tl
6333     \keys_set:no { nicematrix / RulesBis } \l_@@_other_keys_tl
6334     \bool_if:NTF \l_@@_dotted_bool
6335         \@@_hline_iv:
6336     {
6337         \tl_if_empty:NTF \l_@@_tikz_rule_tl
6338             \@@_hline_iii:
6339             \@@_hline_v:
6340     }
6341 }

```

First the case of a standard rule (without the keys dotted and tikz).

```

6342 \cs_new_protected:Npn \@@_hline_iii:
6343 {
6344     \pgfpicture
6345     \pgfrememberpicturepositiononpagetrue
6346     \pgf@relevantforpicturesizefalse
6347     \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6348     \dim_set_eq:NN \l_tmpa_dim \pgf@x
6349     \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6350     \dim_set:Nn \l_tmpb_dim
6351     {
6352         \pgf@y
6353         - 0.5 \l_@@_rule_width_dim
6354         +
6355         ( \arrayrulewidth * \l_@@_multiplicity_int
6356           + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6357     }
6358     \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6359     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
6360     \bool_lazy_all:nT
6361     {
6362         { \int_compare_p:nNn \l_@@_multiplicity_int > \c_one_int }
6363         { \cs_if_exist_p:N \CT@drsc@ }
6364         { ! \tl_if_blank_p:o \CT@drsc@ }
6365     }
6366     {
6367         \group_begin:
6368         \CT@drsc@
6369         \dim_set:Nn \l_@@_tmpd_dim
6370         {
6371             \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
6372             * ( \l_@@_multiplicity_int - 1 )
6373         }
6374         \pgfpathrectanglecorners
6375         { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6376         { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
6377         \pgfusepathqfill
6378         \group_end:
6379     }
6380     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6381     \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
6382     \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6383     {

```

```

6384     \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
6385     \dim_sub:Nn \l_tmpb_dim \doublerulesep
6386     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6387     \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
6388   }
6389   \CT@arc@
6390   \pgfsetlinewidth { 1.1 \arrayrulewidth }
6391   \pgfsetrectcap
6392   \pgfusepathqstroke
6393   \endpgfpicture
6394 }

```

The following code is for the case of a dotted rule (with our system of rounded dots). The aim is that, by standard the dotted line fits between square brackets (`\hline` doesn't).

```

\begin{bNiceMatrix}
1 & 2 & 3 & 4 \\
\hline
1 & 2 & 3 & 4 \\
\hdottedline
1 & 2 & 3 & 4
\end{bNiceMatrix}

```

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \end{bmatrix}.$$

But, if the user uses `margin`, the dotted line extends to have the same width as a `\hline`.

```

\begin{bNiceMatrix}[margin]
1 & 2 & 3 & 4 \\
\hline
1 & 2 & 3 & 4 \\
\hdottedline
1 & 2 & 3 & 4
\end{bNiceMatrix}

```

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \end{bmatrix}.$$

```

6395 \cs_new_protected:Npn \@@_hline_iv:
6396 {
6397   \pgfpicture
6398   \pgfrememberpicturepositiononpagetrue
6399   \pgf@relevantforpicturesizefalse
6400   \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6401   \dim_set:Nn \l_@@_y_initial_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
6402   \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
6403   \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6404   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
6405   \int_compare:nNnT \l_@@_local_start_int = \c_one_int
6406   {
6407     \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
6408     \bool_if:NF \g_@@_delims_bool
6409     { \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.

```

6410     \tl_if_eq:NnF \g_@@_left_delim_tl (
6411       { \dim_add:Nn \l_@@_x_initial_dim { 0.5 \l_@@_xdots_inter_dim } }
6412     )
6413     \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6414     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
6415     \int_compare:nNnT \l_@@_local_end_int = \c_j_col
6416     {
6417       \dim_add:Nn \l_@@_x_final_dim \l_@@_right_margin_dim
6418       \bool_if:NF \g_@@_delims_bool
6419       { \dim_add:Nn \l_@@_x_final_dim \arraycolsep }
6420       \tl_if_eq:NnF \g_@@_right_delim_tl (
6421         { \dim_gsub:Nn \l_@@_x_final_dim { 0.5 \l_@@_xdots_inter_dim } }
6422       )
6423     }
6424     \CT@arc@
6425     \@@_draw_line:

```

```

6425 \endpgfpicture
6426 }

```

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`).

```

6427 \cs_new_protected:Npn \@@_hline_v:
6428 {
6429   \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).

```

6430   \CT@arc@
6431   \tl_if_empty:NF \l_@@_rule_color_tl
6432   { \tl_put_right:Ne \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6433   \pgfrememberpicturepositiononpagetrue
6434   \pgf@relevantforpicturesizefalse
6435   \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6436   \dim_set_eq:NN \l_tmpa_dim \pgf@x
6437   \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6438   \dim_set:Nn \l_tmpb_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
6439   \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6440   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
6441   \exp_args:No \tikzset \l_@@_tikz_rule_tl
6442   \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6443   ( \l_tmpa_dim , \l_tmpb_dim ) --
6444   ( \l_@@_tmpc_dim , \l_tmpb_dim ) ;
6445   \end { tikzpicture }
6446 }

```

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).

```

6447 \cs_new_protected:Npn \@@_draw_hlines:
6448 {
6449   \int_step_inline:nnn
6450   { \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool 2 1 }
6451   {
6452     \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool
6453     \c@iRow
6454     { \int_eval:n { \c@iRow + 1 } }
6455   }
6456   {
6457     \tl_if_eq:NNF \l_@@_hlines_clist \c_@@_all_tl
6458     { \clist_if_in:NnT \l_@@_hlines_clist { ##1 } }
6459     { \@@_hline:n { position = ##1 , total-width = \arrayrulewidth } }
6460   }
6461 }

```

The command `\@@_Hline:` will be linked to `\Hline` in the environments of `nicematrix`.

```

6462 \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.

```

6463 \cs_set:Npn \@@_Hline_i:n #1
6464 {
6465   \peek_remove_spaces:n
6466   {
6467     \peek_meaning:NNTF \Hline
6468     { \@@_Hline_ii:nn { #1 + 1 } }
6469     { \@@_Hline_iii:n { #1 } }
6470   }
6471 }

```



```

6472 \cs_set:Npn \@@_Hline_ii:nn #1 #2 { \@@_Hline_i:n { #1 } }
6473 \cs_set:Npn \@@_Hline_iii:n #1
6474 { \@@_collect_options:n { \@@_Hline_iv:nn { #1 } } }
6475 \cs_set:Npn \@@_Hline_iv:nn #1 #2
6476 {
6477   \@@_compute_rule_width:n { multiplicity = #1 , #2 }
6478   \skip_vertical:N \l_@@_rule_width_dim
6479   \tl_gput_right:Ne \g_@@_pre_code_after_tl
6480   {
6481     \@@_hline:n
6482     {
6483       multiplicity = #1 ,
6484       position = \int_eval:n { \c@iRow + 1 } ,
6485       total-width = \dim_use:N \l_@@_rule_width_dim ,
6486       #2
6487     }
6488   }
6489   \egroup
6490 }

```

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`).

```

6491 \cs_new_protected:Npn \@@_custom_line:n #1
6492 {
6493   \str_clear_new:N \l_@@_command_str
6494   \str_clear_new:N \l_@@_ccommand_str
6495   \str_clear_new:N \l_@@_letter_str
6496   \tl_clear_new:N \l_@@_other_keys_tl
6497   \keys_set:known { nicematrix / custom-line } { #1 } \l_@@_other_keys_tl

```

If the final user only wants to draw horizontal rules, he does not need to specify a letter (for the vertical rules in the preamble of the array). On the other hand, if he only wants to draw vertical rules, he does not need to define a command (which is the tool to draw horizontal rules in the array). Of course, a definition of custom lines with no letter and no command would be point-less.

```

6498   \bool_lazy_all:nTF
6499   {
6500     { \str_if_empty_p:N \l_@@_letter_str }
6501     { \str_if_empty_p:N \l_@@_command_str }
6502     { \str_if_empty_p:N \l_@@_ccommand_str }
6503   }
6504   { \@@_error:n { No~letter~and~no~command } }
6505   { \@@_custom_line_i:o \l_@@_other_keys_tl }
6506 }
6507 \keys_define:nn { nicematrix / custom-line }
6508 {
6509   letter .str_set:N = \l_@@_letter_str ,
6510   letter .value_required:n = true ,
6511   command .str_set:N = \l_@@_command_str ,
6512   command .value_required:n = true ,
6513   ccommand .str_set:N = \l_@@_ccommand_str ,
6514   ccommand .value_required:n = true ,
6515 }
6516 \cs_generate_variant:Nn \@@_custom_line_i:n { o }
6517 \cs_new_protected:Npn \@@_custom_line_i:n #1
6518 {

```

The following flags will be raised when the keys `tikz`, `dotted` and `color` are used (in the `custom-line`).

```

6519 \bool_set_false:N \l_@@_tikz_rule_bool
6520 \bool_set_false:N \l_@@_dotted_rule_bool
6521 \bool_set_false:N \l_@@_color_bool
6522 \keys_set:nn { nicematrix / custom-line-bis } { #1 }
6523 \bool_if:NT \l_@@_tikz_rule_bool
6524 {
6525   \IfPackageLoadedF { tikz }
6526   { \@@_error:n { tikz~in~custom-line~without~tikz } }
6527   \bool_if:NT \l_@@_color_bool
6528   { \@@_error:n { color~in~custom-line~with~tikz } }
6529 }
6530 \bool_if:NT \l_@@_dotted_rule_bool
6531 {
6532   \int_compare:nNt \l_@@_multiplicity_int > \c_one_int
6533   { \@@_error:n { key~multiplicity~with~dotted } }
6534 }
6535 \str_if_empty:NF \l_@@_letter_str
6536 {
6537   \int_compare:nTF { \str_count:N \l_@@_letter_str != 1 }
6538   { \@@_error:n { Several~letters } }
6539   {
6540     \tl_if_in:NoTF
6541     \c_@@_forbidden_letters_str
6542     \l_@@_letter_str
6543     { \@@_error:ne { Forbidden~letter } \l_@@_letter_str }
6544     {

```

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.

```

6545 \cs_set:cpn { @@ _ \l_@@_letter_str } ##1
6546 { \@@_v_custom_line:n { #1 } }
6547 }
6548 }
6549 }
6550 \str_if_empty:NF \l_@@_command_str { \@@_h_custom_line:n { #1 } }
6551 \str_if_empty:NF \l_@@_ccommand_str { \@@_c_custom_line:n { #1 } }
6552 }
6553 \tl_const:Nn \c_@@_forbidden_letters_tl { lcrpmbVX|()[]!@<> }
6554 \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.

```

6555 \keys_define:nn { nicematrix / custom-line-bis }
6556 {
6557   multiplicity .int_set:N = \l_@@_multiplicity_int ,
6558   multiplicity .initial:n = 1 ,
6559   multiplicity .value_required:n = true ,
6560   color .code:n = \bool_set_true:N \l_@@_color_bool ,
6561   color .value_required:n = true ,
6562   tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6563   tikz .value_required:n = true ,
6564   dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6565   dotted .value_forbidden:n = true ,
6566   total-width .code:n = { } ,
6567   total-width .value_required:n = true ,
6568   width .code:n = { } ,
6569   width .value_required:n = true ,

```

```

6570     sep-color .code:n = { } ,
6571     sep-color .value_required:n = true ,
6572     unknown .code:n = \@@_error:n { Unknown~key~for~custom~line }
6573 }

```

The following keys will indicate whether the keys `dotted`, `tikz` and `color` are used in the use of a `custom-line`.

```

6574 \bool_new:N \l_@@_dotted_rule_bool
6575 \bool_new:N \l_@@_tikz_rule_bool
6576 \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`.

```

6577 \keys_define:nn { nicematrix / custom-line-width }
6578 {
6579     multiplicity .int_set:N = \l_@@_multiplicity_int ,
6580     multiplicity .initial:n = 1 ,
6581     multiplicity .value_required:n = true ,
6582     tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6583     total-width .code:n = \dim_set:Nn \l_@@_rule_width_dim { #1 }
6584                     \bool_set_true:N \l_@@_total_width_bool ,
6585     total-width .value_required:n = true ,
6586     width .meta:n = { total-width = #1 } ,
6587     dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6588 }

```

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`).

```

6589 \cs_new_protected:Npn \@@_h_custom_line:n #1
6590 {

```

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`).

```

6591     \cs_set:cpn { nicematrix - \l_@@_command_str } { \Hline [ #1 ] }
6592     \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_command_str
6593 }

```

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`).

```

6594 \cs_new_protected:Npn \@@_c_custom_line:n #1
6595 {

```

Here, we need an expandable command since it begins with an `\noalign`.

```

6596     \exp_args:Nc \NewExpandableDocumentCommand
6597     { nicematrix - \l_@@_ccommand_str }
6598     { 0 { } m }
6599     {
6600         \noalign
6601         {
6602             \@@_compute_rule_width:n { #1 , ##1 }
6603             \skip_vertical:n { \l_@@_rule_width_dim }
6604             \clist_map_inline:nn
6605             { ##2 }
6606             { \@@_c_custom_line_i:nn { #1 , ##1 } { #####1 } }
6607         }
6608     }
6609     \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_ccommand_str
6610 }

```

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*.

```

6611 \cs_new_protected:Npn \@@_c_custom_line_i:nn #1 #2
6612 {
6613   \str_if_in:nnTF { #2 } { - }
6614   { \@@_cut_on_hyphen:w #2 \q_stop }
6615   { \@@_cut_on_hyphen:w #2 - #2 \q_stop }
6616   \tl_gput_right:Ne \g_@@_pre_code_after_tl
6617   {
6618     \@@_hline:n
6619     {
6620       #1 ,
6621       start = \l_tmpa_tl ,
6622       end = \l_tmpb_tl ,
6623       position = \int_eval:n { \c@iRow + 1 } ,
6624       total-width = \dim_use:N \l_@@_rule_width_dim
6625     }
6626   }
6627 }

6628 \cs_new_protected:Npn \@@_compute_rule_width:n #1
6629 {
6630   \bool_set_false:N \l_@@_tikz_rule_bool
6631   \bool_set_false:N \l_@@_total_width_bool
6632   \bool_set_false:N \l_@@_dotted_rule_bool
6633   \keys_set_known:nn { nicematrix / custom-line-width } { #1 }
6634   \bool_if:NF \l_@@_total_width_bool
6635   {
6636     \bool_if:NTF \l_@@_dotted_rule_bool
6637     { \dim_set:Nn \l_@@_rule_width_dim { 2 \l_@@_xdots_radius_dim } }
6638     {
6639       \bool_if:NF \l_@@_tikz_rule_bool
6640       {
6641         \dim_set:Nn \l_@@_rule_width_dim
6642         {
6643           \arrayrulewidth * \l_@@_multiplicity_int
6644           + \doublerulesep * ( \l_@@_multiplicity_int - 1 )
6645         }
6646       }
6647     }
6648   }
6649 }

6650 \cs_new_protected:Npn \@@_v_custom_line:n #1
6651 {
6652   \@@_compute_rule_width:n { #1 }

```

In the following line, the `\dim_use:N` is mandatory since we do an expansion.

```

6653   \tl_gput_right:Ne \g_@@_array_preamble_tl
6654   { \exp_not:N ! { \skip_horizontal:n { \dim_use:N \l_@@_rule_width_dim } } }
6655   \tl_gput_right:Ne \g_@@_pre_code_after_tl
6656   {
6657     \@@_vline:n
6658     {
6659       #1 ,
6660       position = \int_eval:n { \c@jCol + 1 } ,
6661       total-width = \dim_use:N \l_@@_rule_width_dim
6662     }
6663   }
6664   \@@_rec_preamble:n
6665 }

6666 \@@_custom_line:n
6667 { 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.

```
6668 \cs_new_protected:Npn \@@_test_hline_in_block:nnnnn #1 #2 #3 #4 #5
6669 {
6670   \int_compare:nNnT \l_tmpa_tl > { #1 }
6671   {
6672     \int_compare:nNnT \l_tmpa_tl < { #3 + 1 }
6673     {
6674       \int_compare:nNnT \l_tmpb_tl > { #2 - 1 }
6675       {
6676         \int_compare:nNnT \l_tmpb_tl < { #4 + 1 }
6677         { \bool_gset_false:N \g_tmpa_bool }
6678       }
6679     }
6680   }
6681 }
```

The same for vertical rules.

```
6682 \cs_new_protected:Npn \@@_test_vline_in_block:nnnnn #1 #2 #3 #4 #5
6683 {
6684   \int_compare:nNnT \l_tmpa_tl > { #1 - 1 }
6685   {
6686     \int_compare:nNnT \l_tmpa_tl < { #3 + 1 }
6687     {
6688       \int_compare:nNnT \l_tmpb_tl > { #2 }
6689       {
6690         \int_compare:nNnT \l_tmpb_tl < { #4 + 1 }
6691         { \bool_gset_false:N \g_tmpa_bool }
6692       }
6693     }
6694   }
6695 }

6696 \cs_new_protected:Npn \@@_test_hline_in_stroken_block:nnnn #1 #2 #3 #4
6697 {
6698   \int_compare:nNnT \l_tmpb_tl > { #2 - 1 }
6699   {
6700     \int_compare:nNnT \l_tmpb_tl < { #4 + 1 }
6701     {
6702       \int_compare:nNnTF \l_tmpa_tl = { #1 }
6703       { \bool_gset_false:N \g_tmpa_bool }
6704       {
6705         \int_compare:nNnT \l_tmpa_tl = { #3 + 1 }
6706         { \bool_gset_false:N \g_tmpa_bool }
6707       }
6708     }
6709   }
6710 }

6711 \cs_new_protected:Npn \@@_test_vline_in_stroken_block:nnnn #1 #2 #3 #4
6712 {
6713   \int_compare:nNnT \l_tmpa_tl > { #1 - 1 }
6714   {
6715     \int_compare:nNnT \l_tmpa_tl < { #3 + 1 }
6716     {
6717       \int_compare:nNnTF \l_tmpb_tl = { #2 }
6718       { \bool_gset_false:N \g_tmpa_bool }
6719       {
6720         \int_compare:nNnT \l_tmpb_tl = { #4 + 1 }
6721         { \bool_gset_false:N \g_tmpa_bool }
6722       }
6723     }
6724   }
6725 }
```

```

6723     }
6724   }
6725 }

```

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.

```

6726 \cs_new_protected:Npn \@@_compute_corners:
6727 {

```

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.

```

6728   \seq_clear_new:N \l_@@_corners_cells_seq
6729   \clist_map_inline:Nn \l_@@_corners_clist
6730   {
6731     \str_case:nnF { ##1 }
6732     {
6733       { NW }
6734       { \@@_compute_a_corner:nnnnnn 1 1 1 \c@iRow \c@jCol }
6735       { NE }
6736       { \@@_compute_a_corner:nnnnnn 1 \c@jCol 1 { -1 } \c@iRow 1 }
6737       { SW }
6738       { \@@_compute_a_corner:nnnnnn \c@iRow 1 { -1 } 1 1 \c@jCol }
6739       { SE }
6740       { \@@_compute_a_corner:nnnnnn \c@iRow \c@jCol { -1 } { -1 } 1 1 }
6741     }
6742     { \@@_error:nn { bad-corner } { ##1 } }
6743   }

```

Even if the user has used the key `corners` the list of cells in the corners may be empty.

```

6744   \seq_if_empty:NF \l_@@_corners_cells_seq
6745   {

```

You write on the `aux` file the list of the cells which are in the (empty) corners because you need that information in the `\CodeBefore` since the commands which colors the `rows`, `columns` and `cells` must not color the cells in the corners.

```

6746     \tl_gput_right:Ne \g_@@_aux_tl
6747     {
6748       \seq_set_from_clist:Nn \exp_not:N \l_@@_corners_cells_seq
6749       { \seq_use:Nnnn \l_@@_corners_cells_seq , , , }
6750     }
6751   }
6752 }

```

“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.

```

6753 \cs_new_protected:Npn \@@_compute_a_corner:nnnnnn #1 #2 #3 #4 #5 #6
6754 {

```

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.

```

6755 \bool_set_false:N \l_tmpa_bool
6756 \int_zero_new:N \l_@@_last_empty_row_int
6757 \int_set:Nn \l_@@_last_empty_row_int { #1 }
6758 \int_step_inline:nnnn { #1 } { #3 } { #5 }
6759 {
6760   \@@_test_if_cell_in_a_block:nn { ##1 } { \int_eval:n { #2 } }
6761   \bool_lazy_or:nnTF
6762   {
6763     \cs_if_exist_p:c
6764     { pgf @ sh @ ns @ \@@_env: - ##1 - \int_eval:n { #2 } }
6765   }
6766   \l_tmpb_bool
6767   { \bool_set_true:N \l_tmpa_bool }
6768   {
6769     \bool_if:NF \l_tmpa_bool
6770     { \int_set:Nn \l_@@_last_empty_row_int { ##1 } }
6771   }
6772 }

```

Now, you determine the last empty cell in the row of number 1.

```

6773 \bool_set_false:N \l_tmpa_bool
6774 \int_zero_new:N \l_@@_last_empty_column_int
6775 \int_set:Nn \l_@@_last_empty_column_int { #2 }
6776 \int_step_inline:nnnn { #2 } { #4 } { #6 }
6777 {
6778   \@@_test_if_cell_in_a_block:nn { \int_eval:n { #1 } } { ##1 }
6779   \bool_lazy_or:nnTF
6780   \l_tmpb_bool
6781   {
6782     \cs_if_exist_p:c
6783     { pgf @ sh @ ns @ \@@_env: - \int_eval:n { #1 } - ##1 }
6784   }
6785   { \bool_set_true:N \l_tmpa_bool }
6786   {
6787     \bool_if:NF \l_tmpa_bool
6788     { \int_set:Nn \l_@@_last_empty_column_int { ##1 } }
6789   }
6790 }

```

Now, we loop over the rows.

```

6791 \int_step_inline:nnnn { #1 } { #3 } \l_@@_last_empty_row_int
6792 {

```

We treat the row number `##1` with another loop.

```

6793 \bool_set_false:N \l_tmpa_bool
6794 \int_step_inline:nnnn { #2 } { #4 } \l_@@_last_empty_column_int
6795 {
6796   \@@_test_if_cell_in_a_block:nn { ##1 } { #####1 }
6797   \bool_lazy_or:nnTF
6798   \l_tmpb_bool
6799   {
6800     \cs_if_exist_p:c
6801     { pgf @ sh @ ns @ \@@_env: - ##1 - #####1 }
6802   }
6803   { \bool_set_true:N \l_tmpa_bool }
6804   {
6805     \bool_if:NF \l_tmpa_bool
6806     {
6807       \int_set:Nn \l_@@_last_empty_column_int { #####1 }

```

```

6808         \seq_put_right:Nn
6809         \l_@@_corners_cells_seq
6810         { ##1 - #####1 }
6811     }
6812 }
6813 }
6814 }
6815 }

```

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`).

```

6816 \cs_new_protected:Npn \@@_test_if_cell_in_a_block:nn #1 #2
6817 {
6818     \int_set:Nn \l_tmpa_int { #1 }
6819     \int_set:Nn \l_tmpb_int { #2 }
6820     \bool_set_false:N \l_tmpb_bool
6821     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6822     { \@@_test_if_cell_in_block:nnnnnnn \l_tmpa_int \l_tmpb_int ##1 }
6823 }
6824 \cs_set_protected:Npn \@@_test_if_cell_in_block:nnnnnnn #1 #2 #3 #4 #5 #6 #7
6825 {
6826     \int_compare:nNnF { #3 } > { #1 }
6827     {
6828         \int_compare:nNnF { #1 } > { #5 }
6829         {
6830             \int_compare:nNnF { #4 } > { #2 }
6831             {
6832                 \int_compare:nNnF { #2 } > { #6 }
6833                 { \bool_set_true:N \l_tmpb_bool }
6834             }
6835         }
6836     }
6837 }

```

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.

```

6838 \bool_new:N \l_@@_block_auto_columns_width_bool

```

Up to now, there is only one option available for the environment `{NiceMatrixBlock}`.

```

6839 \keys_define:nn { nicematrix / NiceMatrixBlock }
6840 {
6841     auto-columns-width .code:n =
6842     {
6843         \bool_set_true:N \l_@@_block_auto_columns_width_bool
6844         \dim_gzero_new:N \g_@@_max_cell_width_dim
6845         \bool_set_true:N \l_@@_auto_columns_width_bool
6846     }
6847 }
6848 \NewDocumentEnvironment { NiceMatrixBlock } { ! 0 { } }
6849 {
6850     \int_gincr:N \g_@@_NiceMatrixBlock_int
6851     \dim_zero:N \l_@@_columns_width_dim

```



```

6852 \keys_set:nn { nicematrix / NiceMatrixBlock } { #1 }
6853 \bool_if:NT \l_@@_block_auto_columns_width_bool
6854 {
6855   \cs_if_exist:cT
6856   { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
6857   {
6858     \dim_set:Nn \l_@@_columns_width_dim
6859     {
6860       \use:c
6861       { @@_max_cell_width _ \int_use:N \g_@@_NiceMatrixBlock_int }
6862     }
6863   }
6864 }
6865 }

```

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).

```

6866 {
6867   \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.

```

6868   { \int_gdecr:N \g_@@_NiceMatrixBlock_int }
6869   {
6870     \bool_if:NT \l_@@_block_auto_columns_width_bool
6871     {
6872       \iow_shipout:Nn \@mainaux \ExplSyntaxOn
6873       \iow_shipout:Ne \@mainaux
6874       {
6875         \cs_gset:cpn
6876         { @@ _ 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.

```

6877         { \dim_eval:n { \g_@@_max_cell_width_dim + \arrayrulewidth } }
6878       }
6879       \iow_shipout:Nn \@mainaux \ExplSyntaxOff
6880     }
6881   }
6882   \ignorespacesafterend
6883 }

```

26 The extra nodes

The following command is called in \@@_use_arraybox_with_notes_c: just before the construction of the blocks (if the creation of medium nodes is required, medium nodes are also created for the blocks and that construction uses the standard medium nodes).

```

6884 \cs_new_protected:Npn \@@_create_extra_nodes:
6885 {
6886   \bool_if:nTF \l_@@_medium_nodes_bool
6887   {
6888     \bool_if:NTF \l_@@_large_nodes_bool
6889     \@@_create_medium_and_large_nodes:
6890     \@@_create_medium_nodes:
6891   }
6892   { \bool_if:NT \l_@@_large_nodes_bool \@@_create_large_nodes: }
6893 }

```

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`.

```

6894 \cs_new_protected:Npn \@@_computations_for_medium_nodes:
6895 {
6896   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6897   {
6898     \dim_zero_new:c { l_@@_row_\@@_i: _min_dim }
6899     \dim_set_eq:cN { l_@@_row_\@@_i: _min_dim } \c_max_dim
6900     \dim_zero_new:c { l_@@_row_\@@_i: _max_dim }
6901     \dim_set:cn { l_@@_row_\@@_i: _max_dim } { - \c_max_dim }
6902   }
6903   \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
6904   {
6905     \dim_zero_new:c { l_@@_column_\@@_j: _min_dim }
6906     \dim_set_eq:cN { l_@@_column_\@@_j: _min_dim } \c_max_dim
6907     \dim_zero_new:c { l_@@_column_\@@_j: _max_dim }
6908     \dim_set:cn { l_@@_column_\@@_j: _max_dim } { - \c_max_dim }
6909   }

```

We begin the two nested loops over the rows and the columns of the array.

```

6910   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6911   {
6912     \int_step_variable:nnNn
6913     \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.

```

6914     {
6915       \cs_if_exist:cT
6916       { 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`.

```

6917     {
6918       \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { south-west }
6919       \dim_set:cn { l_@@_row_\@@_i: _min_dim }
6920       { \dim_min:vn { l_@@_row _ \@@_i: _min_dim } \pgf@y }
6921       \seq_if_in:NxF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
6922       {
6923         \dim_set:cn { l_@@_column _ \@@_j: _min_dim }
6924         { \dim_min:vn { l_@@_column _ \@@_j: _min_dim } \pgf@x }
6925       }

```

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`.

```

6926       \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { north-east }
6927       \dim_set:cn { l_@@_row _ \@@_i: _ max_dim }
6928       { \dim_max:vn { l_@@_row _ \@@_i: _ max_dim } \pgf@y }

```

```

6929         \seq_if_in:NcF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
6930         {
6931             \dim_set:cn { l_@@_column _ \@@_j: _ max_dim }
6932             { \dim_max:vn { l_@@_column _ \@@_j: _ max_dim } \pgf@x }
6933         }
6934     }
6935 }
6936 }

```

Now, we have to deal with empty rows or empty columns since we don't have created nodes in such rows and columns.

```

6937 \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6938 {
6939     \dim_compare:nNnT
6940     { \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } } = \c_max_dim
6941     {
6942         \@@_qpoint:n { row - \@@_i: - base }
6943         \dim_set:cn { l_@@_row _ \@@_i: _ max _ dim } \pgf@y
6944         \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim } \pgf@y
6945     }
6946 }
6947 \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
6948 {
6949     \dim_compare:nNnT
6950     { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } } = \c_max_dim
6951     {
6952         \@@_qpoint:n { col - \@@_j: }
6953         \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim } \pgf@y
6954         \dim_set:cn { l_@@_column _ \@@_j: _ min _ dim } \pgf@y
6955     }
6956 }
6957 }

```

Here is the command `\@@_create_medium_nodes:`. When this command is used, the “medium nodes” are created.

```

6958 \cs_new_protected:Npn \@@_create_medium_nodes:
6959 {
6960     \pgfpicture
6961     \pgfrememberpicturepositiononpagetrue
6962     \pgf@relevantforpicturesizefalse
6963     \@@_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”.

```

6964     \cs_set_nopar:Npn \l_@@_suffix_tl { -medium }
6965     \@@_create_nodes:
6966     \endpgfpicture
6967 }

```

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:`.

```

6968 \cs_new_protected:Npn \@@_create_large_nodes:
6969 {
6970     \pgfpicture
6971     \pgfrememberpicturepositiononpagetrue
6972     \pgf@relevantforpicturesizefalse
6973     \@@_computations_for_medium_nodes:

```

¹⁴If we want to create both, we have to use `\@@_create_medium_and_large_nodes:`

```

6974 \@@_computations_for_large_nodes:
6975 \cs_set_nopar:Npn \l_@@_suffix_tl { - large }
6976 \@@_create_nodes:
6977 \endpgfpicture
6978 }
6979 \cs_new_protected:Npn \@@_create_medium_and_large_nodes:
6980 {
6981 \pgfpicture
6982 \pgfrememberpicturepositiononpagetrue
6983 \pgf@relevantforpicturesizefalse
6984 \@@_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”.

```

6985 \cs_set_nopar:Npn \l_@@_suffix_tl { - medium }
6986 \@@_create_nodes:
6987 \@@_computations_for_large_nodes:
6988 \cs_set_nopar:Npn \l_@@_suffix_tl { - large }
6989 \@@_create_nodes:
6990 \endpgfpicture
6991 }

```

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.

```

6992 \cs_new_protected:Npn \@@_computations_for_large_nodes:
6993 {
6994 \int_set_eq:NN \l_@@_first_row_int \c_one_int
6995 \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`.

```

6996 \int_step_variable:nNn { \c@iRow - 1 } \@@_i:
6997 {
6998 \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim }
6999 {
7000 (
7001 \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } +
7002 \dim_use:c { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
7003 )
7004 / 2
7005 }
7006 \dim_set_eq:cc { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
7007 { l_@@_row _ \@@_i: _ min _ dim }
7008 }
7009 \int_step_variable:nNn { \c@jCol - 1 } \@@_j:
7010 {
7011 \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim }
7012 {
7013 (
7014 \dim_use:c { l_@@_column _ \@@_j: _ max _ dim } +
7015 \dim_use:c
7016 { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
7017 )
7018 / 2
7019 }
7020 \dim_set_eq:cc { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
7021 { l_@@_column _ \@@_j: _ max _ dim }
7022 }

```

Here, we have to use `\dim_sub:cn` because of the number 1 in the name.

```

7023 \dim_sub:cn
7024 { l_@@_column _ 1 _ min _ dim }
7025 \l_@@_left_margin_dim

```

```

7026 \dim_add:cn
7027 { l_@@_column _ \int_use:N \c@jCol _ max _ dim }
7028 \l_@@_right_margin_dim
7029 }

```

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).

```

7030 \cs_new_protected:Npn \@@_create_nodes:
7031 {
7032   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7033   {
7034     \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
7035     {

```

We draw the rectangular node for the cell (`\@@_i-\@@_j`).

```

7036       \@@_pgf_rect_node:nnnnn
7037       { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7038       { \dim_use:c { l_@@_column_ \@@_j: _min_dim } }
7039       { \dim_use:c { l_@@_row_ \@@_i: _min_dim } }
7040       { \dim_use:c { l_@@_column_ \@@_j: _max_dim } }
7041       { \dim_use:c { l_@@_row_ \@@_i: _max_dim } }
7042     \str_if_empty:NF \l_@@_name_str
7043     {
7044       \pgfnodealias
7045       { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
7046       { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7047     }
7048   }
7049 }

```

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 .

```

7050   \seq_map_pairwise_function:NNN
7051   \g_@@_multicolumn_cells_seq
7052   \g_@@_multicolumn_sizes_seq
7053   \@@_node_for_multicolumn:nn
7054 }

```

```

7055 \cs_new_protected:Npn \@@_extract_coords_values: #1 - #2 \q_stop
7056 {
7057   \cs_set_nopar:Npn \@@_i: { #1 }
7058   \cs_set_nopar:Npn \@@_j: { #2 }
7059 }

```

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”).

```

7060 \cs_new_protected:Npn \@@_node_for_multicolumn:nn #1 #2
7061 {
7062   \@@_extract_coords_values: #1 \q_stop
7063   \@@_pgf_rect_node:nnnnn
7064   { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7065   { \dim_use:c { l_@@_column _ \@@_j: _min _ dim } }
7066   { \dim_use:c { l_@@_row _ \@@_i: _min _ dim } }
7067   { \dim_use:c { l_@@_column _ \int_eval:n { \@@_j: +#2-1 } _ max _ dim } }
7068   { \dim_use:c { l_@@_row _ \@@_i: _ max _ dim } }
7069   \str_if_empty:NF \l_@@_name_str

```

```

7070 {
7071   \pgfnodealias
7072   { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
7073   { \int_use:N \g_@@_env_int - \@@_i: - \@@_j: \l_@@_suffix_tl}
7074 }
7075 }

```

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).

```

7076 \keys_define:nn { nicematrix / Block / FirstPass }
7077 {
7078   j .code:n = \str_set:Nn \l_@@_hpos_block_str j
7079     \bool_set_true:N \l_@@_p_block_bool ,
7080   j .value_forbidden:n = true ,
7081   l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7082   l .value_forbidden:n = true ,
7083   r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7084   r .value_forbidden:n = true ,
7085   c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7086   c .value_forbidden:n = true ,
7087   L .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7088   L .value_forbidden:n = true ,
7089   R .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7090   R .value_forbidden:n = true ,
7091   C .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7092   C .value_forbidden:n = true ,
7093   t .code:n = \str_set:Nn \l_@@_vpos_block_str t ,
7094   t .value_forbidden:n = true ,
7095   T .code:n = \str_set:Nn \l_@@_vpos_block_str T ,
7096   T .value_forbidden:n = true ,
7097   b .code:n = \str_set:Nn \l_@@_vpos_block_str b ,
7098   b .value_forbidden:n = true ,
7099   B .code:n = \str_set:Nn \l_@@_vpos_block_str B ,
7100   B .value_forbidden:n = true ,
7101   m .code:n = \str_set:Nn \l_@@_vpos_block_str c ,
7102   m .value_forbidden:n = true ,
7103   v-center .meta:n = m ,
7104   p .code:n = \bool_set_true:N \l_@@_p_block_bool ,
7105   p .value_forbidden:n = true ,
7106   color .code:n =
7107     \@@_color:n { #1 }
7108     \tl_set_rescan:Nnn
7109       \l_@@_draw_tl
7110       { \char_set_catcode_other:N ! }
7111       { #1 } ,
7112   color .value_required:n = true ,
7113   respect-arraystretch .code:n =
7114     \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
7115   respect-arraystretch .value_forbidden:n = true ,
7116 }

```

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.

```
7117 \cs_new_protected:Npn \@@_Block: { \@@_collect_options:n { \@@_Block_i: } }
```

```
7118 \NewExpandableDocumentCommand \@@_Block_i: { m m D < > { } +m }
7119 {
```

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).

```
7120   \peek_remove_spaces:n
7121   {
7122     \tl_if_blank:nTF { #2 }
7123     { \@@_Block_ii:nnnnn \c_one_int \c_one_int }
7124     {
7125       \int_compare:nNnTF { \char_value_catcode:n { 45 } } = { 13 }
7126       \@@_Block_i_czech \@@_Block_i
7127       #2 \q_stop
7128     }
7129     { #1 } { #3 } { #4 }
7130   }
7131 }
```

With the following construction, we extract the values of i and j in the first mandatory argument of the command.

```
7132 \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`.

```
7133 {
7134   \char_set_catcode_active:N -
7135   \cs_new:Npn \@@_Block_i_czech #1-#2 \q_stop { \@@_Block_ii:nnnnn { #1 } { #2 } }
7136 }
```

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.

```
7137 \cs_new_protected:Npn \@@_Block_ii:nnnnn #1 #2 #3 #4 #5
7138 {
```

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).

```
7139   \bool_lazy_or:nnTF
7140   { \tl_if_blank_p:n { #1 } }
7141   { \str_if_eq_p:on \c_@@_star_str { #1 } }
7142   { \int_set:Nn \l_tmpa_int { 100 } }
7143   { \int_set:Nn \l_tmpa_int { #1 } }
7144   \bool_lazy_or:nnTF
7145   { \tl_if_blank_p:n { #2 } }
7146   { \str_if_eq_p:on \c_@@_star_str { #2 } }
7147   { \int_set:Nn \l_tmpb_int { 100 } }
7148   { \int_set:Nn \l_tmpb_int { #2 } }
```

If the block is mono-column.

```
7149   \int_compare:nNnTF \l_tmpb_int = \c_one_int
7150   {
7151     \tl_if_empty:NnTF \l_@@_hpos_cell_tl
```

```

7152      { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_c_str }
7153      { \str_set:NV \l_@@_hpos_block_str \l_@@_hpos_cell_tl }
7154    }
7155    { \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.

```

7156    \keys_set_known:nn { nicematrix / Block / FirstPass } { #3 }
7157    \tl_set:Nx \l_tmpa_tl
7158    {
7159      { \int_use:N \c@iRow }
7160      { \int_use:N \c@jCol }
7161      { \int_eval:n { \c@iRow + \l_tmpa_int - 1 } }
7162      { \int_eval:n { \c@jCol + \l_tmpb_int - 1 } }
7163    }

```

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).

```

7164    \bool_set_false:N \l_tmpa_bool
7165    \bool_if:NT \l_@@_amp_in_blocks_bool
7166    \tl_if_in:nnT is faster than \str_if_in:nnT.
7167    { \tl_if_in:nnT { #5 } { & } { \bool_set_true:N \l_tmpa_bool } }
7168    \bool_case:nF
7169    {
7170      \l_tmpa_bool { \@@_Block_vii:eennn }
7171      \l_@@_p_block_bool { \@@_Block_vi:eennn }

```

For the blocks mono-column, we will compose right now in a box in order to compute its width and take that width into account for the width of the column. However, if the column is a `X` column, we should not do that since the width is determined by another way. This should be the same for the `p`, `m` and `b` columns and we should modify that point. However, for the `X` column, it’s imperative. Otherwise, the process for the determination of the widths of the columns will be wrong.

```

7171    \l_@@_X_bool { \@@_Block_v:eennn }
7172    { \tl_if_empty_p:n { #5 } } { \@@_Block_v:eennn }
7173    { \int_compare_p:nNn \l_tmpa_int = \c_one_int } { \@@_Block_iv:eennn }
7174    { \int_compare_p:nNn \l_tmpb_int = \c_one_int } { \@@_Block_iv:eennn }
7175    }
7176    { \@@_Block_v:eennn }
7177    { \l_tmpa_int } { \l_tmpb_int } { #3 } { #4 } { #5 }
7178  }

```

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.

```

7179 \cs_generate_variant:Nn \@@_Block_iv:nnnnn { e e }
7180 \cs_new_protected:Npn \@@_Block_iv:nnnnn #1 #2 #3 #4 #5
7181 {
7182   \int_gincr:N \g_@@_block_box_int
7183   \cs_set_protected_nopar:Npn \diagbox ##1 ##2
7184   {

```



```

7185 \tl_gput_right:Ne \g_@@_pre_code_after_tl
7186 {
7187     \@@_actually_diagbox:nnnnnn
7188     { \int_use:N \c@iRow }
7189     { \int_use:N \c@jCol }
7190     { \int_eval:n { \c@iRow + #1 - 1 } }
7191     { \int_eval:n { \c@jCol + #2 - 1 } }
7192     { \g_@@_row_style_tl \exp_not:n { ##1 } }
7193     { \g_@@_row_style_tl \exp_not:n { ##2 } }
7194 }
7195 }
7196 \box_gclear_new:c
7197 { \g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }

```

Now, we will actually compose the content of the `\Block` in a TeX box. *Be careful*: if after the construction of the box, the boolean `\g_@@_rotate_bool` is raised (which means that the command `\rotate` was present in the content of the `\Block`) we will rotate the box but also, maybe, change the position of the baseline!

```

7198 \hbox_gset:cn
7199 { \g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
7200 {

```

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}`).

```

7201 \tl_if_empty:NTF \l_@@_color_tl
7202 { \int_compare:nNnT { #2 } = \c_one_int \set@color }
7203 { \@@_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`.

```

7204 \int_compare:nNnT { #1 } = \c_one_int
7205 {
7206     \int_if_zero:nTF \c@iRow
7207     {

```

In the following code, the value of `code-for-first-row` contains a `\Block` (in order to have the “first row” centered). But, that block will be executed, since it is entirely contained in the first row, the value of `code-for-first-row` will be inserted once again... with the same command `\Block`. That’s why we have to nullify the command `\Block`.

```

$\begin{bNiceMatrix}%
[
  r,
  first-row,
  last-col,
  code-for-first-row = \Block{}{\scriptstyle\color{blue} \arabic{jCol}},
  code-for-last-col = \scriptstyle \color{blue} \arabic{iRow}
]
& & & & \\
-2 & 3 & -4 & 5 & \\
3 & -4 & 5 & -6 & \\
-4 & 5 & -6 & 7 & \\
5 & -6 & 7 & -8 & \\
\end{bNiceMatrix}$

```

```

7208 \cs_set_eq:NN \Block \@@_NullBlock:
7209 \l_@@_code_for_first_row_tl
7210 }
7211 {

```

```

7212         \int_compare:nNnT \c@iRow = \l_@@_last_row_int
7213         {
7214             \cs_set_eq:NN \Block \@@_NullBlock:
7215             \l_@@_code_for_last_row_tl
7216         }
7217     }
7218     \g_@@_row_style_tl
7219 }

```

The following command will be no-op when `respect-arraystretch` is in force.

```

7220 \@@_reset_arraystretch:
7221 \dim_zero:N \extrarowheight

```

#4 is the optional argument of the command `\Block`, provided with the syntax `<...>`.

```

7222 #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.).

```

7223 \@@_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}`.

```

7224 \bool_if:NTF \l_@@_tabular_bool
7225 {
7226     \bool_lazy_all:nTF
7227     {
7228         { \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.

```

7229         { ! \dim_compare_p:nNn \l_@@_col_width_dim < \c_zero_dim }
7230         { ! \g_@@_rotate_bool }
7231     }

```

When the block is mono-column in a column with a fixed width (e.g. `p{3cm}`), we use a `{minipage}`.

```

7232 {
7233     \use:e
7234     {

```

The `\exp_not:N` is mandatory before `\begin`.

```

7235         \exp_not:N \begin { minipage }%
7236         [ \str_lowercase:o \l_@@_vpos_block_str ]
7237         { \l_@@_col_width_dim }
7238         \str_case:on \l_@@_hpos_block_str
7239         { c \centering r \raggedleft l \raggedright }
7240     }
7241     #5
7242     \end { minipage }
7243 }

```

In the other cases, we use a `{tabular}`.

```

7244 {
7245     \use:e
7246     {
7247         \exp_not:N \begin { tabular }%
7248         [ \str_lowercase:o \l_@@_vpos_block_str ]
7249         { @ { } \l_@@_hpos_block_str @ { } }
7250     }
7251     #5
7252     \end { tabular }
7253 }
7254 }

```

If we are in a mathematical array (`\l_@@_tabular_bool` is `false`). The composition is always done with an `{array}` (never with a `{minipage}`).

```

7255     {
7256         \c_math_toggle_token
7257         \use:e
7258         {
7259             \exp_not:N \begin { array }%
7260             [ \str_lowercase:o \l_@@_vpos_block_str ]
7261             { @ { } \l_@@_hpos_block_str @ { } }
7262         }
7263         #5
7264         \end { array }
7265         \c_math_toggle_token
7266     }
7267 }

```

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).

```

7268 \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.

```

7269 \int_compare:nNnT { #2 } = \c_one_int
7270 {
7271     \dim_gset:Nn \g_@@_blocks_wd_dim
7272     {
7273         \dim_max:nn
7274         \g_@@_blocks_wd_dim
7275         {
7276             \box_wd:c
7277             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7278         }
7279     }
7280 }

```

If we are in a mono-row block we take into account the height and the depth of that block for the height and the depth of the row, excepted when the block uses explicitly an option of vertical position.

```

7281 \bool_lazy_and:nnT
7282 { \int_compare_p:nNn { #1 } = \c_one_int }

```

If the user has not used a key for the vertical position of the block, then `\l_@@_vpos_block_str` remains empty.

```

7283 { \str_if_empty_p:N \l_@@_vpos_block_str }
7284 {
7285     \dim_gset:Nn \g_@@_blocks_ht_dim
7286     {
7287         \dim_max:nn
7288         \g_@@_blocks_ht_dim
7289         {
7290             \box_ht:c
7291             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7292         }
7293     }
7294     \dim_gset:Nn \g_@@_blocks_dp_dim
7295     {
7296         \dim_max:nn
7297         \g_@@_blocks_dp_dim
7298         {
7299             \box_dp:c
7300             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7301         }

```

```

7302     }
7303   }
7304   \seq_gput_right:Ne \g_@@_blocks_seq
7305   {
7306     \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.

```

7307   {
7308     \exp_not:n { #3 } ,
7309     \l_@@_hpos_block_str ,

```

Now, we put a key for the vertical alignment.

```

7310     \bool_if:NT \g_@@_rotate_bool
7311     {
7312       \bool_if:NTF \g_@@_rotate_c_bool
7313       { m }
7314       { \int_compare:nNnT \c@iRow = \l_@@_last_row_int T }
7315     }
7316   }
7317   {
7318     \box_use_drop:c
7319     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7320   }
7321 }
7322 \bool_set_false:N \g_@@_rotate_c_bool
7323 }

```

```

7324 \cs_new:Npn \@@_adjust_hpos_rotate:
7325 {
7326   \bool_if:NT \g_@@_rotate_bool
7327   {
7328     \str_set:Ne \l_@@_hpos_block_str
7329     {
7330       \bool_if:NTF \g_@@_rotate_c_bool
7331       { c }
7332       {
7333         \str_case:onF \l_@@_vpos_block_str
7334         { b l B l t r T r }
7335         { \int_compare:nNnTF \c@iRow = \l_@@_last_row_int r l }
7336       }
7337     }
7338   }
7339 }

```

Despite its name the following command rotates the box of the block *but also does vertical adjustment of the baseline of the block*.

```

7340 \cs_new_protected:Npn \@@_rotate_box_of_block:
7341 {
7342   \box_grotate:cn
7343   { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7344   { 90 }
7345   \int_compare:nNnT \c@iRow = \l_@@_last_row_int
7346   {
7347     \vbox_gset_top:cn
7348     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7349     {
7350       \skip_vertical:n { 0.8 ex }
7351       \box_use:c
7352       { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }

```

```

7353     }
7354   }
7355   \bool_if:NT \g_@@_rotate_c_bool
7356   {
7357     \hbox_gset:cn
7358     { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
7359     {
7360       \c_math_toggle_token
7361       \vcenter
7362       {
7363         \box_use:c
7364         { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
7365       }
7366       \c_math_toggle_token
7367     }
7368   }
7369 }

```

The following macro is for the standard case, where the block is not mono-row and not mono-column and does not use the key p). In that case, the content of the block is *not* composed right now in a box. The composition in a box will be done further, just after the construction of the array (cf. `\@@_draw_blocks:` and above all `\@@_Block_v:nnnnn`).

#1 is i (the number of rows of the block), #2 is j (the number of columns of the block), #3 is the list of *key=values* pairs, #4 are the tokens to put before the math mode and before the composition of the block and #5 is the label (=content) of the block.

```

7370 \cs_generate_variant:Nn \@@_Block_v:nnnnn { e e }
7371 \cs_new_protected:Npn \@@_Block_v:nnnnn #1 #2 #3 #4 #5
7372 {
7373   \seq_gput_right:Ne \g_@@_blocks_seq
7374   {
7375     \l_tmpa_tl
7376     { \exp_not:n { #3 } }
7377     {
7378       \bool_if:NTF \l_@@_tabular_bool
7379       {
7380         \group_begin:

```

The following command will be no-op when `respect-arraystretch` is in force.

```

7381 \@@_reset_arraystretch:
7382 \exp_not:n
7383 {
7384   \dim_zero:N \extrarowheight
7385   #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).

```

7386   \bool_if:NT \c_@@_testphase_table_bool
7387   { \tag_stop:n { table } }
7388   \use:e
7389   {
7390     \exp_not:N \begin { tabular } [ \l_@@_vpos_block_str ]
7391     { @ { } \l_@@_hpos_block_str @ { } }
7392   }
7393   #5
7394   \end { tabular }
7395 }
7396 \group_end:
7397 }

```

When we are *not* in an environment `{NiceTabular}` (or similar).

```

7398     {
7399         \group_begin:
The following will be no-op when respect-arraystretch is in force.
7400         \@@_reset_arraystretch:
7401         \exp_not:n
7402         {
7403             \dim_zero:N \extrarowheight
7404             #4
7405             \c_math_toggle_token
7406             \use:e
7407             {
7408                 \exp_not:N \begin { array } [ \l_@@_vpos_block_str ]
7409                 { @ { } \l_@@_hpos_block_str @ { } }
7410             }
7411             #5
7412             \end { array }
7413             \c_math_toggle_token
7414         }
7415         \group_end:
7416     }
7417 }
7418 }
7419 }

```

The following macro is for the case of a `\Block` which uses the key `p`.

```

7420 \cs_generate_variant:Nn \@@_Block_vi:nnnnn { e e }
7421 \cs_new_protected:Npn \@@_Block_vi:nnnnn #1 #2 #3 #4 #5
7422 {
7423     \seq_gput_right:Ne \g_@@_blocks_seq
7424     {
7425         \l_tmpa_tl
7426         { \exp_not:n { #3 } }
7427         {
7428             \group_begin:
7429             \exp_not:n { #4 #5 }
7430             \group_end:
7431         }
7432     }
7433 }

```

The following macro is for the case of a `\Block` which uses the key `p`.

```

7434 \cs_generate_variant:Nn \@@_Block_vii:nnnnn { e e }
7435 \cs_new_protected:Npn \@@_Block_vii:nnnnn #1 #2 #3 #4 #5
7436 {
7437     \seq_gput_right:Ne \g_@@_blocks_seq
7438     {
7439         \l_tmpa_tl
7440         { \exp_not:n { #3 } }
7441         { \exp_not:n { #4 #5 } }
7442     }
7443 }

```

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).

```

7444 \keys_define:nn { nicematrix / Block / SecondPass }
7445 {
7446     ampersand-in-blocks .bool_set:N = \l_@@_amp_in_blocks_bool ,
7447     ampersand-in-blocks .default:n = true ,
7448     &-in-blocks .meta:n = ampersand-in-blocks ,

```

The sequence `\l_@@_tikz_seq` will contain a sequence of comma-separated lists of keys.

```

7449   tikz .code:n =
7450       \IfPackageLoadedTF { tikz }
7451       { \seq_put_right:Nn \l_@@_tikz_seq { { #1 } } }
7452       { \@@_error:n { tikz~key~without~tikz } } ,
7453   tikz .value_required:n = true ,
7454   fill .code:n =
7455       \tl_set_rescan:Nnn
7456       \l_@@_fill_tl
7457       { \char_set_catcode_other:N ! }
7458       { #1 } ,
7459   fill .value_required:n = true ,
7460   opacity .tl_set:N = \l_@@_opacity_tl ,
7461   opacity .value_required:n = true ,
7462   draw .code:n =
7463       \tl_set_rescan:Nnn
7464       \l_@@_draw_tl
7465       { \char_set_catcode_other:N ! }
7466       { #1 } ,
7467   draw .default:n = default ,
7468   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
7469   rounded-corners .default:n = 4 pt ,
7470   color .code:n =
7471       \@@_color:n { #1 }
7472       \tl_set_rescan:Nnn
7473       \l_@@_draw_tl
7474       { \char_set_catcode_other:N ! }
7475       { #1 } ,
7476   borders .clist_set:N = \l_@@_borders_clist ,
7477   borders .value_required:n = true ,
7478   hvlines .meta:n = { vlines , hlines } ,
7479   vlines .bool_set:N = \l_@@_vlines_block_bool ,
7480   vlines .default:n = true ,
7481   hlines .bool_set:N = \l_@@_hlines_block_bool ,
7482   hlines .default:n = true ,
7483   line-width .dim_set:N = \l_@@_line_width_dim ,
7484   line-width .value_required:n = true ,

```

Some keys have not a property `.value_required:n` (or similar) because they are in `FirstPass`.

```

7485   j .code:n = \str_set:Nn \l_@@_hpos_block_str j
7486       \bool_set_true:N \l_@@_p_block_bool ,
7487   l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7488   r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7489   c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7490   L .code:n = \str_set:Nn \l_@@_hpos_block_str l
7491       \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7492   R .code:n = \str_set:Nn \l_@@_hpos_block_str r
7493       \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7494   C .code:n = \str_set:Nn \l_@@_hpos_block_str c
7495       \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7496   t .code:n = \str_set:Nn \l_@@_vpos_block_str t ,
7497   T .code:n = \str_set:Nn \l_@@_vpos_block_str T ,
7498   b .code:n = \str_set:Nn \l_@@_vpos_block_str b ,
7499   B .code:n = \str_set:Nn \l_@@_vpos_block_str B ,
7500   m .code:n = \str_set:Nn \l_@@_vpos_block_str c ,
7501   m .value_forbidden:n = true ,
7502   v-center .meta:n = m ,
7503   p .code:n = \bool_set_true:N \l_@@_p_block_bool ,
7504   p .value_forbidden:n = true ,
7505   name .tl_set:N = \l_@@_block_name_str ,
7506   name .value_required:n = true ,
7507   name .initial:n = ,
7508   respect-arraystretch .code:n =
7509       \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,

```

```

7510     respect-arraystretch .value_forbidden:n = true ,
7511     transparent .bool_set:N = \l_@@_transparent_bool ,
7512     transparent .default:n = true ,
7513     transparent .initial:n = false ,
7514     unknown .code:n = \@@_error:n { Unknown~key~for~Block }
7515 }

```

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.

```

7516 \cs_new_protected:Npn \@@_draw_blocks:
7517 {
7518     \bool_if:NTF \c_@@_tagging_array_bool
7519     { \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign: }
7520     { \cs_set_eq:NN \ialign \@@_old_ialign: }
7521     \seq_map_inline:Nn \g_@@_blocks_seq { \@@_Block_iv:nnnnnn ##1 }
7522 }
7523 \cs_generate_variant:Nn \@@_Block_v:nnnnnn { n n V V }
7524 \cs_new_protected:Npn \@@_Block_iv:nnnnnn #1 #2 #3 #4 #5 #6
7525 {

```

The integer `\l_@@_last_row_int` will be the last row of the block and `\l_@@_last_col_int` its last column.

```

7526     \int_zero_new:N \l_@@_last_row_int
7527     \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.

```

7528     \int_compare:nNnTF { #3 } > { 99 }
7529     { \int_set_eq:NN \l_@@_last_row_int \c@iRow }
7530     { \int_set:Nn \l_@@_last_row_int { #3 } }
7531     \int_compare:nNnTF { #4 } > { 99 }
7532     { \int_set_eq:NN \l_@@_last_col_int \c@jCol }
7533     { \int_set:Nn \l_@@_last_col_int { #4 } }
7534     \int_compare:nNnTF \l_@@_last_col_int > \g_@@_col_total_int
7535     {
7536         \bool_lazy_and:nnTF
7537         \l_@@_preamble_bool
7538         {
7539             \int_compare_p:n
7540             { \l_@@_last_col_int <= \g_@@_static_num_of_col_int }
7541         }
7542         {
7543             \msg_error:nnnn { nicematrix } { Block-too-large-2 } { #1 } { #2 }
7544             \@@_msg_redirect_name:nn { Block-too-large-2 } { none }
7545             \@@_msg_redirect_name:nn { columns-not-used } { none }
7546         }
7547         { \msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 } }
7548     }
7549     {
7550         \int_compare:nNnTF \l_@@_last_row_int > \g_@@_row_total_int
7551         { \msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 } }
7552         {
7553             \@@_Block_v:nnVVnn
7554             { #1 }
7555             { #2 }
7556             \l_@@_last_row_int
7557             \l_@@_last_col_int

```



```

7558         { #5 }
7559         { #6 }
7560     }
7561 }
7562 }

```

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

```

7563 \cs_new_protected:Npn \@@_Block_v:nnnnnn #1 #2 #3 #4 #5 #6
7564 {

```

The group is for the keys.

```

7565     \group_begin:
7566     \int_compare:nNt { #1 } = { #3 }
7567     { \str_set:Nn \l_@@_vpos_block_str { t } }
7568     \keys_set:nn { nicematrix / Block / SecondPass } { #5 }

```

If the content of the block contains `&`, we will have a special treatment (since the cell must be divided in several sub-cells). Remark that `\tl_if_in:nnT` is faster then `\str_if_in:nnT`.

```

7569     \tl_if_in:nnT { #6 } { & } { \bool_set_true:N \l_@@_ampersand_bool }

7570     \bool_lazy_and:nnT
7571     \l_@@_vlines_block_bool
7572     { ! \l_@@_ampersand_bool }
7573     {
7574         \tl_gput_right:Ne \g_nicematrix_code_after_tl
7575         {
7576             \@@_vlines_block:nnn
7577             { \exp_not:n { #5 } }
7578             { #1 - #2 }
7579             { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7580         }
7581     }
7582     \bool_if:NT \l_@@_hlines_block_bool
7583     {
7584         \tl_gput_right:Ne \g_nicematrix_code_after_tl
7585         {
7586             \@@_hlines_block:nnn
7587             { \exp_not:n { #5 } }
7588             { #1 - #2 }
7589             { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7590         }
7591     }
7592     \bool_if:NF \l_@@_transparent_bool
7593     {
7594         \bool_lazy_and:nnF \l_@@_vlines_block_bool \l_@@_hlines_block_bool
7595         {

```

The sequence of the positions of the blocks (excepted the blocks with the key `hvlines`) will be used when drawing the rules (in fact, there is also the `\multicolumn` and the `\diagbox` in that sequence).

```

7596         \seq_gput_left:Ne \g_@@_pos_of_blocks_seq
7597         { { #1 } { #2 } { #3 } { #4 } { \l_@@_block_name_str } }
7598     }
7599 }

```

```

7600 \tl_if_empty:NF \l_@@_draw_tl
7601 {
7602     \bool_lazy_or:nnT \l_@@_hlines_block_bool \l_@@_vlines_block_bool
7603     { \@@_error:n { hlines~with~color } }
7604     \tl_gput_right:Ne \g_nicematrix_code_after_tl
7605     {
7606         \@@_stroke_block:nnn

```

#5 are the options

```

7607         { \exp_not:n { #5 } }
7608         { #1 - #2 }
7609         { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7610     }
7611     \seq_gput_right:Nn \g_@@_pos_of_stroken_blocks_seq
7612     { { #1 } { #2 } { #3 } { #4 } }
7613 }
7614 \clist_if_empty:NF \l_@@_borders_clist
7615 {
7616     \tl_gput_right:Ne \g_nicematrix_code_after_tl
7617     {
7618         \@@_stroke_borders_block:nnn
7619         { \exp_not:n { #5 } }
7620         { #1 - #2 }
7621         { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7622     }
7623 }
7624 \tl_if_empty:NF \l_@@_fill_tl
7625 {
7626     \tl_if_empty:NF \l_@@_opacity_tl
7627     {
7628         \tl_if_head_eq_meaning:nNTF \l_@@_fill_tl [
7629             {
7630                 \tl_set:Ne \l_@@_fill_tl
7631                 {
7632                     [ opacity = \l_@@_opacity_tl ,
7633                     \tl_tail:o \l_@@_fill_tl
7634                 }
7635             }
7636             {
7637                 \tl_set:Ne \l_@@_fill_tl
7638                 { [ opacity = \l_@@_opacity_tl ] { \l_@@_fill_tl } }
7639             }
7640         ]
7641         \tl_gput_right:Ne \g_@@_pre_code_before_tl
7642         {
7643             \exp_not:N \roundedrectanglecolor
7644             \tl_if_head_eq_meaning:oNTF \l_@@_fill_tl [
7645                 { \l_@@_fill_tl }
7646                 { { \l_@@_fill_tl } }
7647             { #1 - #2 }
7648             { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7649             { \dim_use:N \l_@@_rounded_corners_dim }
7650         }
7651     }
7652 \seq_if_empty:NF \l_@@_tikz_seq
7653 {
7654     \tl_gput_right:Ne \g_nicematrix_code_before_tl
7655     {
7656         \@@_block_tikz:nnnnn
7657         { \seq_use:Nn \l_@@_tikz_seq { , } }
7658         { #1 }
7659         { #2 }
7660         { \int_use:N \l_@@_last_row_int }
7661         { \int_use:N \l_@@_last_col_int }
7662     }
7663 }
7664 \cs_set_protected_nopar:Npn \diagbox ##1 ##2

```

We will have in that last field a list of list of Tikz keys.

```

7665 {
7666   \tl_gput_right:N \g_@@_pre_code_after_tl
7667   {
7668     \@@_actually_diagbox:nnnnn
7669     { #1 }
7670     { #2 }
7671     { \int_use:N \l_@@_last_row_int }
7672     { \int_use:N \l_@@_last_col_int }
7673     { \exp_not:n { ##1 } } { \exp_not:n { ##2 } }
7674   }
7675 }

```

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`

		one
		two
three	four	four block
six	seven	five
		eight

We highlight the node `1-1-block-short`

		one
		two
three	four	four block
six	seven	five
		eight

The construction of the node corresponding to the merged cells.

```

7676 \pgfpicture
7677 \pgfrememberpicturepositiononpagetrue
7678 \pgf@relevantforpicturesizefalse
7679 \@@_qpoint:n { row - #1 }
7680 \dim_set_eq:NN \l_tmpa_dim \pgf@y
7681 \@@_qpoint:n { col - #2 }
7682 \dim_set_eq:NN \l_tmpb_dim \pgf@x
7683 \@@_qpoint:n { row - \int_eval:n { \l_@@_last_row_int + 1 } }
7684 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7685 \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7686 \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.

```

7687 \@@_pgf_rect_node:nnnnn
7688 { \@@_env: - #1 - #2 - block }
7689 \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
7690 \str_if_empty:NF \l_@@_block_name_str
7691 {
7692   \pgfnodealias
7693   { \@@_env: - \l_@@_block_name_str }
7694   { \@@_env: - #1 - #2 - block }
7695   \str_if_empty:NF \l_@@_name_str
7696   {
7697     \pgfnodealias
7698     { \l_@@_name_str - \l_@@_block_name_str }
7699     { \@@_env: - #1 - #2 - block }
7700   }
7701 }

```

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.

```

7702   \bool_if:NF \l_@@_hpos_of_block_cap_bool
7703   {
7704     \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.

```

7705     \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
7706     {

```

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.

```

7707       \cs_if_exist:cT
7708       { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
7709       {
7710         \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
7711         {
7712           \pgfpointanchor { \@@_env: - ##1 - #2 } { west }
7713           \dim_set:Nn \l_tmpb_dim { \dim_min:nn \l_tmpb_dim \pgf@x }
7714         }
7715       }
7716     }

```

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.

```

7717     \dim_compare:nNnT \l_tmpb_dim = \c_max_dim
7718     {
7719       \@@_qpoint:n { col - #2 }
7720       \dim_set_eq:NN \l_tmpb_dim \pgf@x
7721     }
7722     \dim_set:Nn \l_@@_tmpd_dim { - \c_max_dim }
7723     \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
7724     {
7725       \cs_if_exist:cT
7726       { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
7727       {
7728         \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
7729         {
7730           \pgfpointanchor
7731           { \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
7732           { east }
7733           \dim_set:Nn \l_@@_tmpd_dim { \dim_max:nn \l_@@_tmpd_dim \pgf@x }
7734         }
7735       }
7736     }
7737     \dim_compare:nNnT \l_@@_tmpd_dim = { - \c_max_dim }
7738     {
7739       \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7740       \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
7741     }
7742     \@@_pgf_rect_node:nnnn
7743     { \@@_env: - #1 - #2 - block - short }
7744     \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
7745   }

```

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.

```

7746   \bool_if:NT \l_@@_medium_nodes_bool
7747   {
7748     \@@_pgf_rect_node:nnn

```

```

7749 { \@@_env: - #1 - #2 - block - medium }
7750 { \pgfpointanchor { \@@_env: - #1 - #2 - medium } { north-west } }
7751 {
7752   \pgfpointanchor
7753   { \@@_env:
7754     - \int_use:N \l_@@_last_row_int
7755     - \int_use:N \l_@@_last_col_int - medium
7756   }
7757   { south-east }
7758 }
7759 }
7760 \endpgfpicture

7761 \bool_if:NTF \l_@@_ampersand_bool
7762 {
7763   \seq_set_split:Nnn \l_tmpa_seq { & } { #6 }
7764   \int_zero_new:N \l_@@_split_int
7765   \int_set:Nn \l_@@_split_int { \seq_count:N \l_tmpa_seq }
7766   \pgfpicture
7767   \pgfrememberpicturepositiononpagetrue
7768   \pgf@relevantforpicturesizefalse
7769   \@@_qpoint:n { row - #1 }
7770   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7771   \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
7772   \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
7773   \@@_qpoint:n { col - #2 }
7774   \dim_set_eq:NN \l_tmpa_dim \pgf@x
7775   \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
7776   \dim_set:Nn \l_tmpb_dim
7777     { ( \pgf@x - \l_tmpa_dim ) / \int_use:N \l_@@_split_int }
7778   \bool_lazy_or:nnT
7779     \l_@@_vlines_block_bool
7780     { \tl_if_eq_p:NN \l_@@_vlines_clist \c_@@_all_tl }
7781   {
7782     \int_step_inline:nn { \l_@@_split_int - 1 }
7783     {
7784       \pgfpathmoveto
7785       {
7786         \pgfpoint
7787         { \l_tmpa_dim + ##1 \l_tmpb_dim }
7788         \l_@@_tmpc_dim
7789       }
7790       \pgfpathlineto
7791       {
7792         \pgfpoint
7793         { \l_tmpa_dim + ##1 \l_tmpb_dim }
7794         \l_@@_tmpd_dim
7795       }
7796       \CT@arc@
7797       \pgfsetlinewidth { 1.1 \arrayrulewidth }
7798       \pgfsetrectcap
7799       \pgfusepathqstroke
7800     }
7801   }
7802   \@@_qpoint:n { row - #1 - base }
7803   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7804   \int_step_inline:nn \l_@@_split_int
7805   {
7806     \group_begin:
7807     \dim_set:Nn \col@sep
7808       { \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
7809     \pgftransformshift
7810     {

```

```

7811 \pgfpoint
7812 {
7813     \str_case:on \l_@@_hpos_block_str
7814     {
7815         l { \l_tmpa_dim + ##1 \l_tmpb_dim - \l_tmpb_dim + \col@sep}
7816         c { \l_tmpa_dim + ##1 \l_tmpb_dim - 0.5 \l_tmpb_dim }
7817         r { \l_tmpa_dim + ##1 \l_tmpb_dim - \col@sep }
7818     }
7819 }
7820 { \l_@@_tmpc_dim }
7821 }
7822 \pgfset
7823 {
7824     inner~xsep = \c_zero_dim ,
7825     inner~ysep = \c_zero_dim
7826 }
7827 \pgfnode
7828 { rectangle }
7829 {
7830     \str_case:on \l_@@_hpos_block_str
7831     {
7832         c { base }
7833         l { base~west }
7834         r { base~east }
7835     }
7836 }
7837 { \seq_item:Nn \l_tmpa_seq { ##1 } } { } { }
7838 \group_end:
7839 }
7840 \endpgfpicture
7841 }

```

Now the case where there is no ampersand & in the content of the block.

```

7842 {
7843     \bool_if:NTF \l_@@_p_block_bool
7844     {

```

When the final user has used the key p, we have to compute the width.

```

7845 \pgfpicture
7846 \pgfrememberpicturepositiononpagetrue
7847 \pgf@relevantforpicturesizefalse
7848 \bool_if:NTF \l_@@_hpos_of_block_cap_bool
7849 {
7850     \@@_qpoint:n { col - #2 }
7851     \dim_gset_eq:NN \g_tmpa_dim \pgf@x
7852     \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7853 }
7854 {
7855     \pgfpointanchor { \@@_env: - #1 - #2 - block - short } { west }
7856     \dim_gset_eq:NN \g_tmpa_dim \pgf@x
7857     \pgfpointanchor { \@@_env: - #1 - #2 - block - short } { east }
7858 }
7859 \dim_gset:Nn \g_tmpb_dim { \pgf@x - \g_tmpa_dim }
7860 \endpgfpicture
7861 \hbox_set:Nn \l_@@_cell_box
7862 {
7863     \begin { minipage } [ \str_lowercase:o \l_@@_vpos_block_str ]
7864     { \g_tmpb_dim }
7865     \str_case:on \l_@@_hpos_block_str
7866     { c \centering r \raggedleft l \raggedright j { } }
7867     #6
7868     \end { minipage }
7869 }
7870 }

```

```

7871 { \hbox_set:Nn \l_@@_cell_box { \set@color #6 } }
7872 \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:

```

Now, we will put the label of the block. We recall that `\l_@@_vpos_block_str` is empty when the user has not used a key for the vertical position of the block.

```

7873 \pgfpicture
7874 \pgfrememberpicturepositiononpagetrue
7875 \pgf@relevantforpicturesizefalse
7876 \bool_lazy_any:nTF
7877 {
7878   { \str_if_empty_p:N \l_@@_vpos_block_str } % added 2024/06/29
7879   { \str_if_eq_p:on \l_@@_vpos_block_str { c } }
7880   { \str_if_eq_p:on \l_@@_vpos_block_str { T } }
7881   { \str_if_eq_p:on \l_@@_vpos_block_str { B } }
7882 }
7883 {

```

If we are in the first column, we must put the block as if it was with the key `r`.

```

7884 \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`.

```

7885 \bool_if:nT \g_@@_last_col_found_bool
7886 {
7887   \int_compare:nNnT { #2 } = \g_@@_col_total_int
7888   { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_l_str }
7889 }

```

`\l_tmpa_tl` will contain the anchor of the PGF node which will be used.

```

7890 \tl_set:Ne \l_tmpa_tl
7891 {
7892   \str_case:on \l_@@_vpos_block_str
7893   {

```

We recall that `\l_@@_vpos_block_str` is empty when the user has not used a key for the vertical position of the block.

```

7894 { } { % added 2024-06-29
7895   \str_case:on \l_@@_hpos_block_str
7896   {
7897     c { center }
7898     l { west }
7899     r { east }
7900     j { center }
7901   }
7902 }
7903 c {
7904   \str_case:on \l_@@_hpos_block_str
7905   {
7906     c { center }
7907     l { west }
7908     r { east }
7909     j { center }
7910   }
7911 }
7912 }
7913 T {
7914   \str_case:on \l_@@_hpos_block_str
7915   {
7916     c { north }
7917     l { north-west }
7918     r { north-east }
7919     j { north }
7920   }
7921 }

```

```

7922     }
7923     B {
7924         \str_case:on \l_@@_hpos_block_str
7925         {
7926             c { south }
7927             l { south-west }
7928             r { south-east }
7929             j { south }
7930         }
7931     }
7932 }
7933 }
7934 }
7935 \pgftransformshift
7936 {
7937     \pgfpointanchor
7938     {
7939         \@@_env: - #1 - #2 - block
7940         \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
7941     }
7942     { \l_tmpa_tl }
7943 }
7944 \pgfset
7945 {
7946     inner-xsep = \c_zero_dim ,
7947     inner-ysep = \c_zero_dim
7948 }
7949 \pgfnode
7950 { rectangle }
7951 { \l_tmpa_tl }
7952 { \box_use_drop:N \l_@@_cell_box } { } { }
7953 }

```

End of the case when $\backslash l_@@_vpos_block_str$ is equal to c, T or B. Now, the other cases.

```

7954 {
7955     \pgfextracty \l_tmpa_dim
7956     {
7957         \@@_qpoint:n
7958         {
7959             row - \str_if_eq:onTF \l_@@_vpos_block_str { b } { #3 } { #1 }
7960             - base
7961         }
7962     }
7963     \dim_sub:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }

```

We retrieve (in $\backslash pgf@x$) the x -value of the center of the block.

```

7964 \pgfpointanchor
7965 {
7966     \@@_env: - #1 - #2 - block
7967     \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
7968 }
7969 {
7970     \str_case:on \l_@@_hpos_block_str
7971     {
7972         c { center }
7973         l { west }
7974         r { east }
7975         j { center }
7976     }
7977 }

```

We put the label of the block which has been composed in $\backslash l_@@_cell_box$.

```

7978 \pgftransformshift { \pgfpoint \pgf@x \l_tmpa_dim }
7979 \pgfset { inner-sep = \c_zero_dim }

```



```

7980     \pgfnode
7981     { rectangle }
7982     {
7983         \str_case:on \l_@@_hpos_block_str
7984         {
7985             c { base }
7986             l { base~west }
7987             r { base~east }
7988             j { base }
7989         }
7990     }
7991     { \box_use_drop:N \l_@@_cell_box } { } { }
7992 }
7993 \endpgfpicture
7994 }
7995 \group_end:
7996 }

```

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).

```

7997 \cs_new_protected:Npn \@@_stroke_block:nnn #1 #2 #3
7998 {
7999     \group_begin:
8000     \tl_clear:N \l_@@_draw_tl
8001     \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8002     \keys_set_known:nn { nicematrix / BlockStroke } { #1 }
8003     \pgfpicture
8004     \pgfrememberpicturerepositiononpagetrue
8005     \pgf@relevantforpicturesizefalse
8006     \tl_if_empty:NF \l_@@_draw_tl
8007     {

```

If the user has used the key `color` of the command `\Block` without value, the color fixed by `\arrayrulecolor` is used.

```

8008     \tl_if_eq:NNTF \l_@@_draw_tl \c_@@_default_tl
8009     { \CT@arc@ }
8010     { \@@_color:o \l_@@_draw_tl }
8011 }
8012 \pgfsetcornersarced
8013 {
8014     \pgfpoint
8015     { \l_@@_rounded_corners_dim }
8016     { \l_@@_rounded_corners_dim }
8017 }
8018 \@@_cut_on_hyphen:w #2 \q_stop
8019 \int_compare:nNnF \l_tmpa_tl > \c@iRow
8020 {
8021     \int_compare:nNnF \l_tmpb_tl > \c@jCol
8022     {
8023         \@@_qpoint:n { row - \l_tmpa_tl }
8024         \dim_set_eq:NN \l_tmpb_dim \pgf@y
8025         \@@_qpoint:n { col - \l_tmpb_tl }
8026         \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
8027         \@@_cut_on_hyphen:w #3 \q_stop
8028         \int_compare:nNnT \l_tmpa_tl > \c@iRow
8029         { \tl_set:No \l_tmpa_tl { \int_use:N \c@iRow } }
8030         \int_compare:nNnT \l_tmpb_tl > \c@jCol
8031         { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
8032         \@@_qpoint:n { row - \int_eval:n { \l_tmpa_tl + 1 } }
8033         \dim_set_eq:NN \l_tmpa_dim \pgf@y
8034         \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
8035         \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x

```

```

8036         \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
8037         \pgfpathrectanglecorners
8038         { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
8039         { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
8040         \dim_compare:nNnTF \l_@@_rounded_corners_dim = \c_zero_dim
8041         { \pgfusepathqstroke }
8042         { \pgfusepath { stroke } }
8043     }
8044 }
8045 \endpgfpicture
8046 \group_end:
8047 }

```

Here is the set of keys for the command `\@@_stroke_block:nnn`.

```

8048 \keys_define:nn { nicematrix / BlockStroke }
8049 {
8050     color .tl_set:N = \l_@@_draw_tl ,
8051     draw .code:n =
8052         \tl_if_empty:eF { #1 } { \tl_set:Nn \l_@@_draw_tl { #1 } } ,
8053     draw .default:n = default ,
8054     line-width .dim_set:N = \l_@@_line_width_dim ,
8055     rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
8056     rounded-corners .default:n = 4 pt
8057 }

```

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).

```

8058 \cs_new_protected:Npn \@@_vlines_block:nnn #1 #2 #3
8059 {
8060     \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8061     \keys_set_known:nn { nicematrix / BlockBorders } { #1 }
8062     \@@_cut_on_hyphen:w #2 \q_stop
8063     \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
8064     \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
8065     \@@_cut_on_hyphen:w #3 \q_stop
8066     \tl_set:Ne \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
8067     \tl_set:Ne \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
8068     \int_step_inline:nnn \l_@@_tmpd_tl \l_tmpb_tl
8069     {
8070         \use:e
8071         {
8072             \@@_vline:n
8073             {
8074                 position = ##1 ,
8075                 start = \l_@@_tmpc_tl ,
8076                 end = \int_eval:n { \l_tmpa_tl - 1 } ,
8077                 total-width = \dim_use:N \l_@@_line_width_dim
8078             }
8079         }
8080     }
8081 }
8082 \cs_new_protected:Npn \@@_hlines_block:nnn #1 #2 #3
8083 {
8084     \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8085     \keys_set_known:nn { nicematrix / BlockBorders } { #1 }
8086     \@@_cut_on_hyphen:w #2 \q_stop
8087     \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
8088     \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
8089     \@@_cut_on_hyphen:w #3 \q_stop
8090     \tl_set:Ne \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
8091     \tl_set:Ne \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
8092     \int_step_inline:nnn \l_@@_tmpc_tl \l_tmpa_tl

```

```

8093 {
8094   \use:e
8095   {
8096     \@@_hline:n
8097     {
8098       position = ##1 ,
8099       start = \l_@@_tmpd_tl ,
8100       end = \int_eval:n { \l_tmpb_tl - 1 } ,
8101       total-width = \dim_use:N \l_@@_line_width_dim
8102     }
8103   }
8104 }
8105 }

```

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).

```

8106 \cs_new_protected:Npn \@@_stroke_borders_block:nnn #1 #2 #3
8107 {
8108   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8109   \keys_set_known:nn { nicematrix / BlockBorders } { #1 }
8110   \dim_compare:nNnTF \l_@@_rounded_corners_dim > \c_zero_dim
8111   { \@@_error:n { borders~forbidden } }
8112   {
8113     \tl_clear_new:N \l_@@_borders_tikz_tl
8114     \keys_set:no
8115     { nicematrix / OnlyForTikzInBorders }
8116     \l_@@_borders_clist
8117     \@@_cut_on_hyphen:w #2 \q_stop
8118     \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
8119     \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
8120     \@@_cut_on_hyphen:w #3 \q_stop
8121     \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
8122     \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
8123     \@@_stroke_borders_block_i:
8124   }
8125 }
8126 \hook_gput_code:nnn { begindocument } { . }
8127 {
8128   \cs_new_protected:Npe \@@_stroke_borders_block_i:
8129   {
8130     \c_@@_pgfortikzpicture_tl
8131     \@@_stroke_borders_block_ii:
8132     \c_@@_endpgfortikzpicture_tl
8133   }
8134 }
8135 \cs_new_protected:Npn \@@_stroke_borders_block_ii:
8136 {
8137   \pgfrememberpicturepositiononpagetrue
8138   \pgf@relevantforpicturesizefalse
8139   \CT@arc@
8140   \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
8141   \clist_if_in:NnT \l_@@_borders_clist { right }
8142   { \@@_stroke_vertical:n \l_tmpb_tl }
8143   \clist_if_in:NnT \l_@@_borders_clist { left }
8144   { \@@_stroke_vertical:n \l_@@_tmpd_tl }
8145   \clist_if_in:NnT \l_@@_borders_clist { bottom }
8146   { \@@_stroke_horizontal:n \l_tmpa_tl }
8147   \clist_if_in:NnT \l_@@_borders_clist { top }
8148   { \@@_stroke_horizontal:n \l_@@_tmpc_tl }
8149 }

```

```

8150 \keys_define:nn { nicematrix / OnlyForTikzInBorders }
8151 {
8152   tikz .code:n =
8153     \cs_if_exist:NTF \tikzpicture
8154     { \tl_set:Nn \l_@@_borders_tikz_tl { #1 } }
8155     { \@@_error:n { tikz~in~borders~without~tikz } } ,
8156   tikz .value_required:n = true ,
8157   top .code:n = ,
8158   bottom .code:n = ,
8159   left .code:n = ,
8160   right .code:n = ,
8161   unknown .code:n = \@@_error:n { bad~border }
8162 }

```

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).

```

8163 \cs_new_protected:Npn \@@_stroke_vertical:n #1
8164 {
8165   \@@_qpoint:n \l_@@_tmpc_tl
8166   \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
8167   \@@_qpoint:n \l_tmpa_tl
8168   \dim_set:Nn \l_@@_tmpc_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
8169   \@@_qpoint:n { #1 }
8170   \tl_if_empty:NTF \l_@@_borders_tikz_tl
8171   {
8172     \pgfpathmoveto { \pgfpoint \pgf@x \l_tmpb_dim }
8173     \pgfpathlineto { \pgfpoint \pgf@x \l_@@_tmpc_dim }
8174     \pgfusepathqstroke
8175   }
8176   {
8177     \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
8178     ( \pgf@x , \l_tmpb_dim ) -- ( \pgf@x , \l_@@_tmpc_dim ) ;
8179   }
8180 }

```

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).

```

8181 \cs_new_protected:Npn \@@_stroke_horizontal:n #1
8182 {
8183   \@@_qpoint:n \l_@@_tmpd_tl
8184   \clist_if_in:NnTF \l_@@_borders_clist { left }
8185   { \dim_set:Nn \l_tmpa_dim { \pgf@x - 0.5 \l_@@_line_width_dim } }
8186   { \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \l_@@_line_width_dim } }
8187   \@@_qpoint:n \l_tmpb_tl
8188   \dim_set:Nn \l_tmpb_dim { \pgf@x + 0.5 \l_@@_line_width_dim }
8189   \@@_qpoint:n { #1 }
8190   \tl_if_empty:NTF \l_@@_borders_tikz_tl
8191   {
8192     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }
8193     \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
8194     \pgfusepathqstroke
8195   }
8196   {
8197     \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
8198     ( \l_tmpa_dim , \pgf@y ) -- ( \l_tmpb_dim , \pgf@y ) ;
8199   }
8200 }

```

Here is the set of keys for the command \@@_stroke_borders_block:nnn.

```

8201 \keys_define:nn { nicematrix / BlockBorders }
8202 {
8203   borders .clist_set:N = \l_@@_borders_clist ,

```

```

8204     rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
8205     rounded-corners .default:n = 4 pt ,
8206     line-width .dim_set:N = \l_@@_line_width_dim
8207 }

```

The following command will be used if the key `tikz` has been used for the command `\Block`.

`#1` is a *list of lists* of Tikz keys used with the path.

Example: `{\offset=1pt,draw,red},{\offset=2pt,draw,blue}}`

which arises from a command such as :

`\Block[tikz={\offset=1pt,draw,red},tikz={\offset=2pt,draw,blue}]{2-2}{}`

The arguments `#2` and `#3` are the coordinates of the first cell and `#4` and `#5` the coordinates of the last cell of the block.

```

8208 \cs_generate_variant:Nn \@@_block_tikz:nnnnn { o }
8209 \cs_new_protected:Npn \@@_block_tikz:nnnnn #1 #2 #3 #4 #5
8210 {
8211     \begin { tikzpicture }
8212     \@@_clip_with_rounded_corners:

```

We use `clist_map_inline:nn` because `#5` is a list of lists.

```

8213     \clist_map_inline:nn { #1 }
8214     {

```

We extract the key `offset` which is *not* a key of TikZ but a key added by `nicematrix`.

```

8215         \keys_set_known:nnN { nicematrix / SpecialOffset } { ##1 } \l_tmpa_tl
8216         \use:e { \exp_not:N \path [ \l_tmpa_tl ] }
8217         (
8218             [
8219                 xshift = \dim_use:N \l_@@_offset_dim ,
8220                 yshift = - \dim_use:N \l_@@_offset_dim
8221             ]
8222             #2 -| #3
8223         )
8224         rectangle
8225         (
8226             [
8227                 xshift = - \dim_use:N \l_@@_offset_dim ,
8228                 yshift = \dim_use:N \l_@@_offset_dim
8229             ]
8230             \int_eval:n { #4 + 1 } -| \int_eval:n { #5 + 1 }
8231         ) ;
8232     }
8233     \end { tikzpicture }
8234 }

```

```

8235 \keys_define:nn { nicematrix / SpecialOffset }
8236 { offset .dim_set:N = \l_@@_offset_dim }

```

In some circumstances, we want to nullify the command `\Block`. In order to reach that goal, we will link the command `\Block` to the following command `\@@_NullBlock`: which has the same syntax as the standard command `\Block` but which is no-op.

```

8237 \cs_new_protected:Npn \@@_NullBlock:
8238 { \@@_collect_options:n { \@@_NullBlock_i: } }
8239 \NewExpandableDocumentCommand \@@_NullBlock_i: { m m D < > { } +m }
8240 { }

```

28 How to draw the dotted lines transparently

```

8241 \cs_set_protected:Npn \@@_renew_matrix:
8242 {
8243   \RenewDocumentEnvironment { pmatrix } { }
8244   { \pNiceMatrix }
8245   { \endpNiceMatrix }
8246   \RenewDocumentEnvironment { vmatrix } { }
8247   { \vNiceMatrix }
8248   { \endvNiceMatrix }
8249   \RenewDocumentEnvironment { Vmatrix } { }
8250   { \VNiceMatrix }
8251   { \endVNiceMatrix }
8252   \RenewDocumentEnvironment { bmatrix } { }
8253   { \bNiceMatrix }
8254   { \endbNiceMatrix }
8255   \RenewDocumentEnvironment { Bmatrix } { }
8256   { \BNiceMatrix }
8257   { \endBNiceMatrix }
8258 }

```

29 Automatic arrays

We will extract some keys and pass the other keys to the environment `{NiceArrayWithDelims}`.

```

8259 \keys_define:nn { nicematrix / Auto }
8260 {
8261   columns-type .tl_set:N = \l_@@_columns_type_tl ,
8262   columns-type .value_required:n = true ,
8263   l .meta:n = { columns-type = l } ,
8264   r .meta:n = { columns-type = r } ,
8265   c .meta:n = { columns-type = c } ,
8266   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8267   delimiters / color .value_required:n = true ,
8268   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
8269   delimiters / max-width .default:n = true ,
8270   delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
8271   delimiters .value_required:n = true ,
8272   rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
8273   rounded-corners .default:n = 4 pt
8274 }
8275 \NewDocumentCommand \AutoNiceMatrixWithDelims
8276 { m m O { } } > { \SplitArgument { 1 } { - } } m O { } m ! O { } }
8277 { \@@_auto_nice_matrix:nnnnnn { #1 } { #2 } #4 { #6 } { #3 , #5 , #7 } }
8278 \cs_new_protected:Npn \@@_auto_nice_matrix:nnnnnn #1 #2 #3 #4 #5 #6
8279 {

```

The group is for the protection of the keys.

```

8280 \group_begin:
8281 \keys_set_known:nnN { nicematrix / Auto } { #6 } \l_tmpa_tl
8282 \use:e
8283 {
8284   \exp_not:N \begin { NiceArrayWithDelims } { #1 } { #2 }
8285   { * { #4 } { \exp_not:o \l_@@_columns_type_tl } }
8286   [ \exp_not:o \l_tmpa_tl ]
8287 }
8288 \int_if_zero:nT \l_@@_first_row_int
8289 {
8290   \int_if_zero:nT \l_@@_first_col_int { & }
8291   \prg_replicate:nn { #4 - 1 } { & }
8292   \int_compare:nNt \l_@@_last_col_int > { -1 } { & } \\

```

```

8293     }
8294     \prg_replicate:nn { #3 }
8295     {
8296         \int_if_zero:nT \l_@@_first_col_int { & }
8297         \prg_replicate:nn { #4 - 1 } { { } #5 & } #5
8298         \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
8299     }
8300     \int_compare:nNnT \l_@@_last_row_int > { -2 }
8301     {
8302         \int_if_zero:nT \l_@@_first_col_int { & }
8303         \prg_replicate:nn { #4 - 1 } { & }
8304         \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
8305     }
8306     \end { NiceArrayWithDelims }
8307     \group_end:
8308 }
8309 \cs_set_protected:Npn \@@_define_com:nnn #1 #2 #3
8310 {
8311     \cs_set_protected:cpn { #1 AutoNiceMatrix }
8312     {
8313         \bool_gset_true:N \g_@@_delims_bool
8314         \str_gset:Nx \g_@@_name_env_str { #1 AutoNiceMatrix }
8315         \AutoNiceMatrixWithDelims { #2 } { #3 }
8316     }
8317 }
8318 \@@_define_com:nnn p ( )
8319 \@@_define_com:nnn b [ ]
8320 \@@_define_com:nnn v | |
8321 \@@_define_com:nnn V \l \l
8322 \@@_define_com:nnn B \{ \}

```

We define also a command `\AutoNiceMatrix` similar to the environment `{NiceMatrix}`.

```

8323 \NewDocumentCommand \AutoNiceMatrix { O { } m O { } m ! O { } }
8324 {
8325     \group_begin:
8326     \bool_gset_false:N \g_@@_delims_bool
8327     \AutoNiceMatrixWithDelims . . { #2 } { #4 } [ #1 , #3 , #5 ]
8328     \group_end:
8329 }

```

30 The redefinition of the command `\dotfill`

```

8330 \cs_set_eq:NN \@@_old_dotfill \dotfill
8331 \cs_new_protected:Npn \@@_dotfill:
8332 {

```

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}`).

```

8333     \@@_old_dotfill
8334     \tl_gput_right:Nn \g_@@_cell_after_hook_tl \@@_dotfill_i:
8335 }

```

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`.

```

8336 \cs_new_protected:Npn \@@_dotfill_i:
8337 { \dim_compare:nNt { \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.

```

8338 \cs_new_protected:Npn \@@_diagbox:nn #1 #2
8339 {
8340   \tl_gput_right:Ne \g_@@_pre_code_after_tl
8341   {
8342     \@@_actually_diagbox:nnnnnn
8343     { \int_use:N \c@iRow }
8344     { \int_use:N \c@jCol }
8345     { \int_use:N \c@iRow }
8346     { \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.

```

8347   { \g_@@_row_style_tl \exp_not:n { #1 } }
8348   { \g_@@_row_style_tl \exp_not:n { #2 } }
8349 }

```

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`.

```

8350   \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
8351   {
8352     { \int_use:N \c@iRow }
8353     { \int_use:N \c@jCol }
8354     { \int_use:N \c@iRow }
8355     { \int_use:N \c@jCol }

```

The last argument is for the name of the block.

```

8356   { }
8357 }
8358 }

```

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.

```

8359 \cs_new_protected:Npn \@@_actually_diagbox:nnnnnn #1 #2 #3 #4 #5 #6
8360 {
8361   \pgfpicture
8362   \pgf@relevantforpicturesizefalse
8363   \pgfrememberpicturepositiononpagetrue
8364   \@@_qpoint:n { row - #1 }
8365   \dim_set_eq:NN \l_tmpa_dim \pgf@y
8366   \@@_qpoint:n { col - #2 }
8367   \dim_set_eq:NN \l_tmpb_dim \pgf@x
8368   \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
8369   \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
8370   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8371   \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
8372   \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
8373   \pgfpathlineto { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
8374   {

```


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.

```

8375 \CT@arc@
8376 \pgfsetroundcap
8377 \pgfusepathqstroke
8378 }
8379 \pgfset { inner~sep = 1 pt }
8380 \pgfscope
8381 \pgftransformshift { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
8382 \pgfnode { rectangle } { south~west }
8383 {
8384 \begin { minipage } { 20 cm }
8385 \@@_math_toggle: #5 \@@_math_toggle:
8386 \end { minipage }
8387 }
8388 { }
8389 { }
8390 \endpgfscope
8391 \pgftransformshift { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
8392 \pgfnode { rectangle } { north~east }
8393 {
8394 \begin { minipage } { 20 cm }
8395 \raggedleft
8396 \@@_math_toggle: #6 \@@_math_toggle:
8397 \end { minipage }
8398 }
8399 { }
8400 { }
8401 \endpgfpicture
8402 }

```

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. ??.

In the environments of `nicematrix`, `\CodeAfter` will be linked to `\@@_CodeAfter:`. That macro must *not* be protected since it begins with `\omit`.

```

8403 \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 `\`.

```

8404 \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`.

```

8405 \cs_new_protected:Npn \@@_CodeAfter_ii:n #1 \end
8406 {
8407 \tl_gput_right:Nn \g_nicematrix_code_after_tl { #1 }
8408 \@@_CodeAfter_iv:n
8409 }

```

We catch the argument of the command `\end` (in `#1`).

```

8410 \cs_new_protected:Npn \@@_CodeAfter_iv:n #1
8411 {

```

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.

```
8412 \str_if_eq:eeTF \@currenvir { #1 }
8413 { \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`.

```
8414 {
8415   \tl_gput_right:Nn \g_nicematrix_code_after_tl { \end { #1 } }
8416   \@@_CodeAfter_ii:n
8417 }
8418 }
```

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.

```
8419 \cs_new_protected:Npn \@@_delimiter:nnn #1 #2 #3
8420 {
8421   \pgfpicture
8422   \pgfrememberpicturepositiononpagetrue
8423   \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.

```
8424 \@@_qpoint:n { row - 1 }
8425 \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
8426 \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
8427 \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).

```
8428 \bool_if:nTF { #3 }
8429 { \dim_set_eq:NN \l_tmpa_dim \c_max_dim }
8430 { \dim_set:Nn \l_tmpa_dim { - \c_max_dim } }
8431 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
8432 {
8433   \cs_if_exist:cT
8434   { pgf @ sh @ ns @ \@_env: - ##1 - #2 }
8435   {
8436     \pgfpointanchor
8437     { \@_env: - ##1 - #2 }
8438     { \bool_if:nTF { #3 } { west } { east } }
8439     \dim_set:Nn \l_tmpa_dim
8440     { \bool_if:nTF { #3 } \dim_min:nn \dim_max:nn \l_tmpa_dim \pgf@x }
8441   }
8442 }
```

Now we can put the delimiter with a node of PGF.

```

8443 \pgfset { inner~sep = \c_zero_dim }
8444 \dim_zero:N \nulldelimiterspace
8445 \pgftransformshift
8446 {
8447   \pgfpoint
8448   { \l_tmpa_dim }
8449   { ( \l_@@_y_initial_dim + \l_@@_y_final_dim + \arrayrulewidth ) / 2 }
8450 }
8451 \pgfnode
8452 { rectangle }
8453 { \bool_if:nTF { #3 } { east } { west } }
8454 {

```

Here is the content of the PGF node, that is to say the delimiter, constructed with its right size.

```

8455 \nullfont
8456 \c_math_toggle_token
8457 \@@_color:o \l_@@_delimiters_color_tl
8458 \bool_if:nTF { #3 } { \left #1 } { \left . }
8459 \vcenter
8460 {
8461   \nullfont
8462   \hrule \@height
8463   \dim_eval:n { \l_@@_y_initial_dim - \l_@@_y_final_dim }
8464   \@depth \c_zero_dim
8465   \@width \c_zero_dim
8466 }
8467 \bool_if:nTF { #3 } { \right . } { \right #1 }
8468 \c_math_toggle_token
8469 }
8470 { }
8471 { }
8472 \endpgfpicture
8473 }

```

34 The command \SubMatrix

```

8474 \keys_define:nn { nicematrix / sub-matrix }
8475 {
8476   extra-height .dim_set:N = \l_@@_submatrix_extra_height_dim ,
8477   extra-height .value_required:n = true ,
8478   left-xshift .dim_set:N = \l_@@_submatrix_left_xshift_dim ,
8479   left-xshift .value_required:n = true ,
8480   right-xshift .dim_set:N = \l_@@_submatrix_right_xshift_dim ,
8481   right-xshift .value_required:n = true ,
8482   xshift .meta:n = { left-xshift = #1, right-xshift = #1 } ,
8483   xshift .value_required:n = true ,
8484   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8485   delimiters / color .value_required:n = true ,
8486   slim .bool_set:N = \l_@@_submatrix_slim_bool ,
8487   slim .default:n = true ,
8488   hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
8489   hlines .default:n = all ,
8490   vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
8491   vlines .default:n = all ,
8492   hvlines .meta:n = { hlines, vlines } ,
8493   hvlines .value_forbidden:n = true
8494 }
8495 \keys_define:nn { nicematrix }
8496 {
8497   SubMatrix .inherit:n = nicematrix / sub-matrix ,

```

```

8498 NiceArray / sub-matrix .inherit:n = nicematrix / sub-matrix ,
8499 pNiceArray / sub-matrix .inherit:n = nicematrix / sub-matrix ,
8500 NiceMatrixOptions / sub-matrix .inherit:n = nicematrix / sub-matrix ,
8501 }

```

The following keys set is for the command `\SubMatrix` itself (not the tuning of `\SubMatrix` that can be done elsewhere).

```

8502 \keys_define:nn { nicematrix / SubMatrix }
8503 {
8504   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8505   delimiters / color .value_required:n = true ,
8506   hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
8507   hlines .default:n = all ,
8508   vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
8509   vlines .default:n = all ,
8510   hvlines .meta:n = { hlines, vlines } ,
8511   hvlines .value_forbidden:n = true ,
8512   name .code:n =
8513     \tl_if_empty:nTF { #1 }
8514     { \@@_error:n { Invalid-name } }
8515     {
8516       \regex_match:nnTF { \A[A-Za-z][A-Za-z0-9]*\Z } { #1 }
8517       {
8518         \seq_if_in:NnTF \g_@@_submatrix_names_seq { #1 }
8519         { \@@_error:nn { Duplicate-name-for-SubMatrix } { #1 } }
8520         {
8521           \str_set:Nn \l_@@_submatrix_name_str { #1 }
8522           \seq_gput_right:Nn \g_@@_submatrix_names_seq { #1 }
8523         }
8524       }
8525       { \@@_error:n { Invalid-name } }
8526     } ,
8527   name .value_required:n = true ,
8528   rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
8529   rules .value_required:n = true ,
8530   code .tl_set:N = \l_@@_code_tl ,
8531   code .value_required:n = true ,
8532   unknown .code:n = \@@_error:n { Unknown-key-for-SubMatrix }
8533 }

8534 \NewDocumentCommand \@@_SubMatrix_in_code_before { m m m m ! O { } }
8535 {
8536   \peek_remove_spaces:n
8537   {
8538     \tl_gput_right:Ne \g_@@_pre_code_after_tl
8539     {
8540       \SubMatrix { #1 } { #2 } { #3 } { #4 }
8541       [
8542         delimiters / color = \l_@@_delimiters_color_tl ,
8543         hlines = \l_@@_submatrix_hlines_clist ,
8544         vlines = \l_@@_submatrix_vlines_clist ,
8545         extra-height = \dim_use:N \l_@@_submatrix_extra_height_dim ,
8546         left-xshift = \dim_use:N \l_@@_submatrix_left_xshift_dim ,
8547         right-xshift = \dim_use:N \l_@@_submatrix_right_xshift_dim ,
8548         slim = \bool_to_str:N \l_@@_submatrix_slim_bool ,
8549         #5
8550       ]
8551     }
8552     \@@_SubMatrix_in_code_before_i { #2 } { #3 }
8553   }
8554 }

8555 \NewDocumentCommand \@@_SubMatrix_in_code_before_i
8556 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }

```

```

8557 { \@@_SubMatrix_in_code_before_i:nnnn #1 #2 }
8558 \cs_new_protected:Npn \@@_SubMatrix_in_code_before_i:nnnn #1 #2 #3 #4
8559 {
8560   \seq_gput_right:Ne \g_@@_submatrix_seq
8561   {

```

We use `\str_if_eq:nnTF` because it is fully expandable (and slightly faster than `\tl_if_eq:nnTF`).

```

8562   { \str_if_eq:nnTF { #1 } { last } { \int_use:N \c@iRow } { #1 } }
8563   { \str_if_eq:nnTF { #2 } { last } { \int_use:N \c@jCol } { #2 } }
8564   { \str_if_eq:nnTF { #3 } { last } { \int_use:N \c@iRow } { #3 } }
8565   { \str_if_eq:nnTF { #4 } { last } { \int_use:N \c@jCol } { #4 } }
8566   }
8567 }

```

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`.

```

8568 \hook_gput_code:nnn { begindocument } { . }
8569 {
8570   \cs_set_nopar:Npn \l_@@_argspec_tl { m m m m 0 { } E { _ ^ } { { } { } } }
8571   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
8572   \exp_args:NNo \NewDocumentCommand \@@_SubMatrix \l_@@_argspec_tl
8573   {
8574     \peek_remove_spaces:n
8575     {
8576       \@@_sub_matrix:nnnnnnn
8577       { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } { #7 }
8578     }
8579   }
8580 }

```

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).

```

8581 \NewDocumentCommand \@@_compute_i_j:nn
8582 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
8583 { \@@_compute_i_j:nnnn #1 #2 }
8584 \cs_new_protected:Npn \@@_compute_i_j:nnnn #1 #2 #3 #4
8585 {
8586   \cs_set_nopar:Npn \l_@@_first_i_tl { #1 }
8587   \cs_set_nopar:Npn \l_@@_first_j_tl { #2 }
8588   \cs_set_nopar:Npn \l_@@_last_i_tl { #3 }
8589   \cs_set_nopar:Npn \l_@@_last_j_tl { #4 }
8590   \tl_if_eq:NnT \l_@@_first_i_tl { last }
8591   { \tl_set:NV \l_@@_first_i_tl \c@iRow }
8592   \tl_if_eq:NnT \l_@@_first_j_tl { last }
8593   { \tl_set:NV \l_@@_first_j_tl \c@jCol }
8594   \tl_if_eq:NnT \l_@@_last_i_tl { last }

```

```

8595     { \tl_set:NV \l_@@_last_i_tl \c@iRow }
8596     \tl_if_eq:NnT \l_@@_last_j_tl { last }
8597     { \tl_set:NV \l_@@_last_j_tl \c@jCol }
8598   }
8599   \cs_new_protected:Npn \@@_sub_matrix:nnnnnnn #1 #2 #3 #4 #5 #6 #7
8600   {
8601     \group_begin:

```

The four following token lists correspond to the position of the \SubMatrix.

```

8602     \@@_compute_i_j:nn { #2 } { #3 }
8603     \int_compare:nNnT \l_@@_first_i_tl = \l_@@_last_i_tl
8604     { \cs_set_nopar:Npn \arraystretch { 1 } }
8605     \bool_lazy_or:nnTF
8606     { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
8607     { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
8608     { \@@_error:nn { Construct-too-large } { \SubMatrix } }
8609     {
8610       \str_clear_new:N \l_@@_submatrix_name_str
8611       \keys_set:nn { nicematrix / SubMatrix } { #5 }
8612       \pgfpicture
8613       \pgfrememberpicturepositiononpagetrue
8614       \pgf@relevantforpicturesizefalse
8615       \pgfset { inner~sep = \c_zero_dim }
8616       \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
8617       \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }

```

The last value of \int_step_inline:nnn is provided by currying.

```

8618     \bool_if:NTF \l_@@_submatrix_slim_bool
8619     { \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl }
8620     { \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int }
8621     {
8622       \cs_if_exist:cT
8623       { \pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
8624       {
8625         \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
8626         \dim_compare:nNnT \pgf@x < \l_@@_x_initial_dim
8627         { \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x }
8628       }
8629       \cs_if_exist:cT
8630       { \pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
8631       {
8632         \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
8633         \dim_compare:nNnT \pgf@x > \l_@@_x_final_dim
8634         { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
8635       }
8636     }
8637     \dim_compare:nNnTF \l_@@_x_initial_dim = \c_max_dim
8638     { \@@_error:nn { Impossible~delimiter } { left } }
8639     {
8640       \dim_compare:nNnTF \l_@@_x_final_dim = { - \c_max_dim }
8641       { \@@_error:nn { Impossible~delimiter } { right } }
8642       { \@@_sub_matrix_i:nnnn { #1 } { #4 } { #6 } { #7 } }
8643     }
8644     \endpgfpicture
8645   }
8646   \group_end:
8647 }

```

#1 is the left delimiter, #2 is the right one, #3 is the subscript and #4 is the superscript.

```

8648   \cs_new_protected:Npn \@@_sub_matrix_i:nnnn #1 #2 #3 #4
8649   {
8650     \@@_qpoint:n { row - \l_@@_first_i_tl - base }
8651     \dim_set:Nn \l_@@_y_initial_dim

```

```

8652 {
8653   \fp_to_dim:n
8654   {
8655     \pgf@y
8656     + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
8657   }
8658 }
8659 \@@_qpoint:n { row - \l_@@_last_i_tl - base }
8660 \dim_set:Nn \l_@@_y_final_dim
8661 { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
8662 \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
8663 {
8664   \cs_if_exist:cT
8665   { pgf @ sh @ ns @ \@@_env: - \l_@@_first_i_tl - ##1 }
8666   {
8667     \pgfpointanchor { \@@_env: - \l_@@_first_i_tl - ##1 } { north }
8668     \dim_set:Nn \l_@@_y_initial_dim
8669     { \dim_max:nn \l_@@_y_initial_dim \pgf@y }
8670   }
8671   \cs_if_exist:cT
8672   { pgf @ sh @ ns @ \@@_env: - \l_@@_last_i_tl - ##1 }
8673   {
8674     \pgfpointanchor { \@@_env: - \l_@@_last_i_tl - ##1 } { south }
8675     \dim_compare:nNnT \pgf@y < \l_@@_y_final_dim
8676     { \dim_set_eq:NN \l_@@_y_final_dim \pgf@y }
8677   }
8678 }
8679 \dim_set:Nn \l_tmpa_dim
8680 {
8681   \l_@@_y_initial_dim - \l_@@_y_final_dim +
8682   \l_@@_submatrix_extra_height_dim - \arrayrulewidth
8683 }
8684 \dim_zero:N \nulldelimiterspace

```

We will draw the rules in the `\SubMatrix`.

```

8685 \group_begin:
8686 \pgfsetlinewidth { 1.1 \arrayrulewidth }
8687 \@@_set_CT@arc@:o \l_@@_rules_color_tl
8688 \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`.

```

8689 \seq_map_inline:Nn \g_@@_cols_vlism_seq
8690 {
8691   \int_compare:nNnT \l_@@_first_j_tl < { ##1 }
8692   {
8693     \int_compare:nNnT
8694     { ##1 } < { \int_eval:n { \l_@@_last_j_tl + 1 } }
8695     {

```

First, we extract the value of the abscissa of the rule we have to draw.

```

8696 \@@_qpoint:n { col - ##1 }
8697 \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
8698 \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
8699 \pgfusepathqstroke
8700 }
8701 }
8702 }

```

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.

```

8703 \tl_if_eq:NNTF \l_@@_submatrix_vlines_clist \c_@@_all_tl

```

```

8704 { \int_step_inline:nn { \l_@@_last_j_tl - \l_@@_first_j_tl } }
8705 { \clist_map_inline:Nn \l_@@_submatrix_vlines_clist }
8706 {
8707   \bool_lazy_and:nnTF
8708     { \int_compare_p:nNn { ##1 } > \c_zero_int }
8709     {
8710       \int_compare_p:nNn
8711         { ##1 } < { \l_@@_last_j_tl - \l_@@_first_j_tl + 1 } }
8712     {
8713       \@@_qpoint:n { col - \int_eval:n { ##1 + \l_@@_first_j_tl } }
8714       \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
8715       \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
8716       \pgfusepathqstroke
8717     }
8718     { \@@_error:nnn { Wrong~line~in~SubMatrix } { vertical } { ##1 } }
8719 }

```

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.

```

8720 \tl_if_eq:NNTF \l_@@_submatrix_hlines_clist \c_@@_all_tl
8721 { \int_step_inline:nn { \l_@@_last_i_tl - \l_@@_first_i_tl } }
8722 { \clist_map_inline:Nn \l_@@_submatrix_hlines_clist }
8723 {
8724   \bool_lazy_and:nnTF
8725     { \int_compare_p:nNn { ##1 } > \c_zero_int }
8726     {
8727       \int_compare_p:nNn
8728         { ##1 } < { \l_@@_last_i_tl - \l_@@_first_i_tl + 1 } }
8729     {
8730       \@@_qpoint:n { row - \int_eval:n { ##1 + \l_@@_first_i_tl } }

```

We use a group to protect `\l_tmpa_dim` and `\l_tmpb_dim`.

```

8731   \group_begin:
We compute in \l_tmpa_dim the x-value of the left end of the rule.
8732   \dim_set:Nn \l_tmpa_dim
8733     { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
8734   \str_case:nn { #1 }
8735   {
8736     ( { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
8737     [ { \dim_sub:Nn \l_tmpa_dim { 0.2 mm } }
8738     \{ { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
8739     }
8740   \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }

```

We compute in `\l_tmpb_dim` the *x*-value of the right end of the rule.

```

8741   \dim_set:Nn \l_tmpb_dim
8742     { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
8743   \str_case:nn { #2 }
8744   {
8745     ) { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
8746     ] { \dim_add:Nn \l_tmpb_dim { 0.2 mm } }
8747     \} { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
8748   }
8749   \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
8750   \pgfusepathqstroke
8751   \group_end:
8752 }
8753 { \@@_error:nnn { Wrong~line~in~SubMatrix } { horizontal } { ##1 } }
8754 }

```

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).

```

8755 \str_if_empty:NF \l_@@_submatrix_name_str

```



```

8756 {
8757   \l_@@_pgf_rect_node:nnnnn \l_@@_submatrix_name_str
8758   \l_@@_x_initial_dim \l_@@_y_initial_dim
8759   \l_@@_x_final_dim \l_@@_y_final_dim
8760 }
8761 \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`.

```

8762 \begin { pgfscope }
8763 \pgftransformshift
8764 {
8765   \pgfpoint
8766   { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
8767   { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
8768 }
8769 \str_if_empty:NTF \l_@@_submatrix_name_str
8770 { \l_@@_node_left:nn #1 { } }
8771 { \l_@@_node_left:nn #1 { \l_@@_env: - \l_@@_submatrix_name_str - left } }
8772 \end { pgfscope }

```

Now, we deal with the right delimiter.

```

8773 \pgftransformshift
8774 {
8775   \pgfpoint
8776   { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
8777   { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
8778 }
8779 \str_if_empty:NTF \l_@@_submatrix_name_str
8780 { \l_@@_node_right:nnnn #2 { } { #3 } { #4 } }
8781 {
8782   \l_@@_node_right:nnnn #2
8783   { \l_@@_env: - \l_@@_submatrix_name_str - right } { #3 } { #4 }
8784 }
8785 \cs_set_eq:NN \pgfpointanchor \l_@@_pgfpointanchor:n
8786 \flag_clear_new:N \l_@@_code_flag
8787 \l_@@_code_tl
8788 }

```

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`.

```

8789 \cs_set_eq:NN \l_@@_old_pgfpointanchor \pgfpointanchor

```

The following command will be linked to `\pgfpointanchor` just before the execution of the option code of the command `\SubMatrix`. In this command, we catch the argument #1 of `\pgfpointanchor` and we apply to it the command `\l_@@_pgfpointanchor_i:nn` before passing it to the original `\pgfpointanchor`. We have to act in an expandable way because the command `\pgfpointanchor` is used in names of Tikz nodes which are computed in an expandable way.

```

8790 \cs_new_protected:Npn \l_@@_pgfpointanchor:n #1
8791 {
8792   \use:e
8793   { \exp_not:N \l_@@_old_pgfpointanchor { \l_@@_pgfpointanchor_i:nn #1 } }
8794 }

```

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 -.

```

8795 \cs_new:Npn \@@_pgfpointanchor_i:nn #1 #2
8796 { #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.

```

8797 \tl_const:Nn \c_@@_integers_alist_tl
8798 {
8799   { 1 } { } { 2 } { } { 3 } { } { 4 } { } { 5 } { }
8800   { 6 } { } { 7 } { } { 8 } { } { 9 } { } { 10 } { }
8801   { 11 } { } { 12 } { } { 13 } { } { 14 } { } { 15 } { }
8802   { 16 } { } { 17 } { } { 18 } { } { 19 } { } { 20 } { }
8803 }

```

```

8804 \cs_new:Npn \@@_pgfpointanchor_ii:w #1-#2\q_stop
8805 {

```

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`.

```

8806   \tl_if_empty:nTF { #2 }
8807   {
8808     \str_case:nVTF { #1 } \c_@@_integers_alist_tl
8809     {
8810       \flag_raise:N \l_@@_code_flag
8811       \int_if_even:nTF { \flag_height:N \l_@@_code_flag }
8812       { \int_eval:n { #1 + \l_@@_first_i_tl - 1 } }
8813       { \int_eval:n { #1 + \l_@@_first_j_tl - 1 } }
8814     }
8815     { #1 }
8816   }

```

If there is an hyphen, we have to see whether we have a node of the form $i-j$, row- i or col- j .

```

8817   { \@@_pgfpointanchor_iii:w { #1 } #2 }
8818 }

```

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`).

```

8819 \cs_new:Npn \@@_pgfpointanchor_iii:w #1 #2 -
8820 {
8821   \str_case:nnF { #1 }
8822   {
8823     { row } { row - \int_eval:n { #2 + \l_@@_first_i_tl - 1 } }
8824     { col } { col - \int_eval:n { #2 + \l_@@_first_j_tl - 1 } }
8825   }

```

Now the case of a node of the form $i-j$.

```

8826   {
8827     \int_eval:n { #1 + \l_@@_first_i_tl - 1 }
8828     - \int_eval:n { #2 + \l_@@_first_j_tl - 1 }
8829   }
8830 }

```

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`).

```

8831 \cs_new_protected:Npn \@@_node_left:nn #1 #2
8832 {
8833   \pgfnode
8834   { rectangle }

```

```

8835 { east }
8836 {
8837   \nullfont
8838   \c_math_toggle_token
8839   \@@_color:o \l_@@_delimiters_color_tl
8840   \left #1
8841   \vcenter
8842   {
8843     \nullfont
8844     \hrule \@height \l_tmpa_dim
8845             \@depth \c_zero_dim
8846             \@width \c_zero_dim
8847   }
8848   \right .
8849   \c_math_toggle_token
8850 }
8851 { #2 }
8852 { }
8853 }

```

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.

```

8854 \cs_new_protected:Npn \@@_node_right:nnnn #1 #2 #3 #4
8855 {
8856   \pgfnode
8857   { rectangle }
8858   { west }
8859   {
8860     \nullfont
8861     \c_math_toggle_token
8862     \colorlet { current-color } { . }
8863     \@@_color:o \l_@@_delimiters_color_tl
8864     \left .
8865     \vcenter
8866     {
8867       \nullfont
8868       \hrule \@height \l_tmpa_dim
8869               \@depth \c_zero_dim
8870               \@width \c_zero_dim
8871     }
8872     \right #1
8873     \tl_if_empty:nF { #3 } { _ { \smash { #3 } } }
8874     ^ { \color { current-color } \smash { #4 } }
8875     \c_math_toggle_token
8876   }
8877   { #2 }
8878   { }
8879 }

```

35 Les commandes `\UnderBrace` et `\OverBrace`

The following commands will be linked to `\UnderBrace` and `\OverBrace` in the `\CodeAfter`.

```

8880 \NewDocumentCommand \@@_UnderBrace { 0 { } m m m 0 { } }
8881 {
8882   \peek_remove_spaces:n
8883   { \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { under } }
8884 }

```

```

8885 \NewDocumentCommand \@@_OverBrace { 0 { } m m 0 { } }
8886 {
8887   \peek_remove_spaces:n
8888   { \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { over } }
8889 }

```

```

8890 \keys_define:nn { nicematrix / Brace }
8891 {
8892   left-shorten .bool_set:N = \l_@@_brace_left_shorten_bool ,
8893   left-shorten .default:n = true ,
8894   left-shorten .value_forbidden:n = true ,
8895   right-shorten .bool_set:N = \l_@@_brace_right_shorten_bool ,
8896   right-shorten .default:n = true ,
8897   right-shorten .value_forbidden:n = true ,
8898   shorten .meta:n = { left-shorten , right-shorten } ,
8899   shorten .value_forbidden:n = true ,
8900   yshift .dim_set:N = \l_@@_brace_yshift_dim ,
8901   yshift .value_required:n = true ,
8902   yshift .initial:n = \c_zero_dim ,
8903   color .tl_set:N = \l_tmpa_tl ,
8904   color .value_required:n = true ,
8905   unknown .code:n = \@@_error:n { Unknown~key~for~Brace }
8906 }

```

#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*.

```

8907 \cs_new_protected:Npn \@@_brace:nnnnn #1 #2 #3 #4 #5
8908 {
8909   \group_begin:

```

The four following token lists correspond to the position of the sub-matrix to which a brace will be attached.

```

8910   \@@_compute_i_j:nn { #1 } { #2 }
8911   \bool_lazy_or:nnTF
8912     { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
8913     { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
8914     {
8915       \str_if_eq:nnTF { #5 } { under }
8916       { \@@_error:nn { Construct-too-large } { \UnderBrace } }
8917       { \@@_error:nn { Construct-too-large } { \OverBrace } }
8918     }
8919     {
8920       \tl_clear:N \l_tmpa_tl
8921       \keys_set:nn { nicematrix / Brace } { #4 }
8922       \tl_if_empty:NF \l_tmpa_tl { \color { \l_tmpa_tl } }
8923       \pgfpicture
8924       \pgfrememberpicturepositiononpagetrue
8925       \pgf@relevantforpicturesizefalse
8926       \bool_if:NT \l_@@_brace_left_shorten_bool
8927       {
8928         \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
8929         \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
8930         {
8931           \cs_if_exist:cT
8932             { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
8933             {
8934               \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
8935
8936               \dim_compare:nNnT \pgf@x < \l_@@_x_initial_dim
8937               { \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x }
8938             }
8939         }
8940       }

```

```

8941 \bool_lazy_or:nnT
8942 { \bool_not_p:n \l_@@_brace_left_shorten_bool }
8943 { \dim_compare_p:nNn \l_@@_x_initial_dim = \c_max_dim }
8944 {
8945   \@@_qpoint:n { col - \l_@@_first_j_tl }
8946   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
8947 }
8948 \bool_if:NT \l_@@_brace_right_shorten_bool
8949 {
8950   \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
8951   \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
8952   {
8953     \cs_if_exist:cT
8954     { \pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
8955     {
8956       \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
8957       \dim_compare:nNnT \pgf@x > \l_@@_x_final_dim
8958       { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
8959     }
8960   }
8961 }
8962 \bool_lazy_or:nnT
8963 { \bool_not_p:n \l_@@_brace_right_shorten_bool }
8964 { \dim_compare_p:nNn \l_@@_x_final_dim = { - \c_max_dim } }
8965 {
8966   \@@_qpoint:n { col - \int_eval:n { \l_@@_last_j_tl + 1 } }
8967   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
8968 }
8969 \pgfset { inner~sep = \c_zero_dim }
8970 \str_if_eq:nnTF { #5 } { under }
8971 { \@@_underbrace_i:n { #3 } }
8972 { \@@_overbrace_i:n { #3 } }
8973 \endpgfpicture
8974 }
8975 \group_end:
8976 }

```

The argument is the text to put above the brace.

```

8977 \cs_new_protected:Npn \@@_overbrace_i:n #1
8978 {
8979   \@@_qpoint:n { row - \l_@@_first_i_tl }
8980   \pgftransformshift
8981   {
8982     \pgfpoint
8983     { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
8984     { \pgf@y + \l_@@_brace_yshift_dim - 3 pt }
8985   }
8986   \pgfnode
8987   { rectangle }
8988   { south }
8989   {
8990     \vtop
8991     {
8992       \group_begin:
8993       \everycr { }
8994       \halign
8995       {
8996         \hfil ## \hfil \crcr
8997         \@@_math_toggle: #1 \@@_math_toggle: \cr
8998         \noalign { \skip_vertical:n { 3 pt } \nointerlineskip }
8999         \c_math_toggle_token
9000         \overbrace
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     }
9009     \group_end:
9010 }
9011 }
9012 { }
9013 { }
9014 }

```

The argument is the text to put under the brace.

```

9015 \cs_new_protected:Npn \@@_underbrace_i:n #1
9016 {
9017     \@@_qpoint:n { row - \int_eval:n { \l_@@_last_i_tl + 1 } }
9018     \pgftransformshift
9019     {
9020         \pgfpoint
9021         { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
9022         { \pgf@y - \l_@@_brace_yshift_dim + 3 pt }
9023     }
9024     \pgfnode
9025     { rectangle }
9026     { north }
9027     {
9028         \group_begin:
9029         \everycr { }
9030         \vbox
9031         {
9032             \halign
9033             {
9034                 \hfil ## \hfil \crrc
9035                 \c_math_toggle_token
9036                 \underbrace
9037                 {
9038                     \hbox_to_wd:nn
9039                     { \l_@@_x_final_dim - \l_@@_x_initial_dim }
9040                     { }
9041                 }
9042                 \c_math_toggle_token
9043                 \cr
9044                 \noalign { \skip_vertical:n { 3 pt } \nointerlineskip }
9045                 \@@_math_toggle: #1 \@@_math_toggle: \cr
9046             }
9047         }
9048         \group_end:
9049     }
9050     { }
9051     { }
9052 }

```

36 The command TikzEveryCell

```

9053 \bool_new:N \l_@@_not_empty_bool
9054 \bool_new:N \l_@@_empty_bool
9055
9056 \keys_define:nn { nicematrix / TikzEveryCell }

```

```

9057 {
9058   not-empty .code:n =
9059     \bool_lazy_or:nnTF
9060     \l_@@_in_code_after_bool
9061     \g_@@_recreate_cell_nodes_bool
9062     { \bool_set_true:N \l_@@_not_empty_bool }
9063     { \@@_error:n { detection-of-empty-cells } } ,
9064   not-empty .value_forbidden:n = true ,
9065   empty .code:n =
9066     \bool_lazy_or:nnTF
9067     \l_@@_in_code_after_bool
9068     \g_@@_recreate_cell_nodes_bool
9069     { \bool_set_true:N \l_@@_empty_bool }
9070     { \@@_error:n { detection-of-empty-cells } } ,
9071   empty .value_forbidden:n = true ,
9072   unknown .code:n = \@@_error:n { Unknown~key~for~TikzEveryCell }
9073 }

```

```

9074
9075
9076 \NewDocumentCommand { \@@_TikzEveryCell } { 0 { } m }
9077 {
9078   \IfPackageLoadedTF { tikz }
9079   {
9080     \group_begin:
9081     \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.

```

9082     \tl_set:Nn \l_tmpa_tl { { #2 } }
9083     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
9084     { \@@_for_a_block:nnnnn ##1 }
9085     \@@_all_the_cells:
9086     \group_end:
9087   }
9088   { \@@_error:n { TikzEveryCell~without~tikz } }
9089 }
9090
9091 \tl_new:N \@@_i_tl
9092 \tl_new:N \@@_j_tl
9093
9094
9095 \cs_new_protected:Nn \@@_all_the_cells:
9096 {
9097   \int_step_variable:nNn \c@iRow \@@_i_tl
9098   {
9099     \int_step_variable:nNn \c@jCol \@@_j_tl
9100     {
9101       \cs_if_exist:cF { cell - \@@_i_tl - \@@_j_tl }
9102       {
9103         \seq_if_in:NeF \l_@@_corners_cells_seq
9104         { \@@_i_tl - \@@_j_tl }
9105         {
9106           \bool_set_false:N \l_tmpa_bool
9107           \cs_if_exist:cTF
9108           { pgf @ sh @ ns @ \@@_env: - \@@_i_tl - \@@_j_tl }
9109           {
9110             \bool_if:NF \l_@@_empty_bool
9111             { \bool_set_true:N \l_tmpa_bool }
9112           }
9113           {
9114             \bool_if:NF \l_@@_not_empty_bool
9115             { \bool_set_true:N \l_tmpa_bool }
9116           }
9117         \bool_if:NT \l_tmpa_bool

```

```

9118         {
9119             \@@_block_tikz:nnnnn
9120             \l_tmpa_tl \@@_i_tl \@@_j_tl \@@_i_tl \@@_j_tl
9121         }
9122     }
9123 }
9124 }
9125 }
9126 }
9127
9128 \cs_new_protected:Nn \@@_for_a_block:nnnnn
9129 {
9130     \bool_if:NF \l_@@_empty_bool
9131     {
9132         \@@_block_tikz:nnnnn
9133         \l_tmpa_tl { #1 } { #2 } { #3 } { #4 }
9134     }
9135     \@@_mark_cells_of_block:nnnn { #1 } { #2 } { #3 } { #4 }
9136 }
9137
9138 \cs_new_protected:Nn \@@_mark_cells_of_block:nnnn
9139 {
9140     \int_step_inline:nnn { #1 } { #3 }
9141     {
9142         \int_step_inline:nnn { #2 } { #4 }
9143         { \cs_set:cpn { cell - ##1 - #####1 } { } }
9144     }
9145 }

```

37 The command \ShowCellNames

```

9146 \NewDocumentCommand \@@_ShowCellNames_CodeBefore { }
9147 {
9148     \dim_gzero_new:N \g_@@_tmpc_dim
9149     \dim_gzero_new:N \g_@@_tmpd_dim
9150     \dim_gzero_new:N \g_@@_tmpe_dim
9151     \int_step_inline:nn \c@iRow
9152     {
9153         \begin { pgfpicture }
9154         \@@_qpoint:n { row - ##1 }
9155         \dim_set_eq:NN \l_tmpa_dim \pgf@y
9156         \@@_qpoint:n { row - \int_eval:n { ##1 + 1 } }
9157         \dim_gset:Nn \g_tmpa_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
9158         \dim_gset:Nn \g_tmpb_dim { \l_tmpa_dim - \pgf@y }
9159         \bool_if:NTF \l_@@_in_code_after_bool
9160         \end { pgfpicture }
9161         \int_step_inline:nn \c@jCol
9162         {
9163             \hbox_set:Nn \l_tmpa_box
9164             { \normalfont \Large \color { red ! 50 } ##1 - #####1 }
9165             \begin { pgfpicture }
9166             \@@_qpoint:n { col - #####1 }
9167             \dim_gset_eq:NN \g_@@_tmpc_dim \pgf@x
9168             \@@_qpoint:n { col - \int_eval:n { #####1 + 1 } }
9169             \dim_gset:Nn \g_@@_tmpd_dim { \pgf@x - \g_@@_tmpc_dim }
9170             \dim_gset_eq:NN \g_@@_tmpe_dim \pgf@x
9171             \endpgfpicture
9172             \end { pgfpicture }
9173             \fp_set:Nn \l_tmpa_fp
9174             {
9175                 \fp_min:nn
9176                 {

```



```

9177         \fp_min:nn
9178         {
9179             \dim_ratio:nn
9180             { \g_@@_tmpd_dim }
9181             { \box_wd:N \l_tmpa_box }
9182         }
9183         {
9184             \dim_ratio:nn
9185             { \g_tmpb_dim }
9186             { \box_ht_plus_dp:N \l_tmpa_box }
9187         }
9188     }
9189     { 1.0 }
9190 }
9191 \box_scale:Nnn \l_tmpa_box
9192 { \fp_use:N \l_tmpa_fp }
9193 { \fp_use:N \l_tmpa_fp }
9194 \pgfpicture
9195 \pgfrememberpicturepositiononpagetrue
9196 \pgf@relevantforpicturesizefalse
9197 \pgftransformshift
9198 {
9199     \pgfpoint
9200     { 0.5 * ( \g_@@_tmpc_dim + \g_@@_tmpe_dim ) }
9201     { \dim_use:N \g_tmpa_dim }
9202 }
9203 \pgfnode
9204 { rectangle }
9205 { center }
9206 { \box_use:N \l_tmpa_box }
9207 { }
9208 { }
9209 \endpgfpicture
9210 }
9211 }
9212 }
9213 \NewDocumentCommand \@@_ShowCellNames { }
9214 {
9215     \bool_if:NT \l_@@_in_code_after_bool
9216     {
9217         \pgfpicture
9218         \pgfrememberpicturepositiononpagetrue
9219         \pgf@relevantforpicturesizefalse
9220         \pgfpathrectanglecorners
9221         { \@@_qpoint:n { 1 } }
9222         {
9223             \@@_qpoint:n
9224             { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
9225         }
9226         \pgfsetfillopacity { 0.75 }
9227         \pgfsetfillcolor { white }
9228         \pgfusepathqfill
9229         \endpgfpicture
9230     }
9231     \dim_gzero_new:N \g_@@_tmpc_dim
9232     \dim_gzero_new:N \g_@@_tmpd_dim
9233     \dim_gzero_new:N \g_@@_tmpe_dim
9234     \int_step_inline:nn \c@iRow
9235     {
9236         \bool_if:NTF \l_@@_in_code_after_bool
9237         {
9238             \pgfpicture
9239             \pgfrememberpicturepositiononpagetrue

```

```

9240     \pgf@relevantforpicturesizefalse
9241   }
9242   { \begin { pgfpicture } }
9243   \@@_qpoint:n { row - ##1 }
9244   \dim_set_eq:NN \l_tmpa_dim \pgf@y
9245   \@@_qpoint:n { row - \int_eval:n { ##1 + 1 } }
9246   \dim_gset:Nn \g_tmpa_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
9247   \dim_gset:Nn \g_tmpb_dim { \l_tmpa_dim - \pgf@y }
9248   \bool_if:NTF \l_@@_in_code_after_bool
9249     { \endpgfpicture }
9250     { \end { pgfpicture } }
9251   \int_step_inline:nn \c@jCol
9252   {
9253     \hbox_set:Nn \l_tmpa_box
9254     {
9255       \normalfont \Large \sffamily \bfseries
9256       \bool_if:NTF \l_@@_in_code_after_bool
9257         { \color { red } }
9258         { \color { red ! 50 } }
9259       ##1 - ####1
9260     }
9261     \bool_if:NTF \l_@@_in_code_after_bool
9262     {
9263       \pgfpicture
9264       \pgfrememberpicturerepositiononpagetrue
9265       \pgf@relevantforpicturesizefalse
9266     }
9267     { \begin { pgfpicture } }
9268     \@@_qpoint:n { col - ####1 }
9269     \dim_gset_eq:NN \g_@@_tmpc_dim \pgf@x
9270     \@@_qpoint:n { col - \int_eval:n { ####1 + 1 } }
9271     \dim_gset:Nn \g_@@_tmpd_dim { \pgf@x - \g_@@_tmpc_dim }
9272     \dim_gset_eq:NN \g_@@_tmpe_dim \pgf@x
9273     \bool_if:NTF \l_@@_in_code_after_bool
9274       { \endpgfpicture }
9275       { \end { pgfpicture } }
9276     \fp_set:Nn \l_tmpa_fp
9277     {
9278       \fp_min:nn
9279       {
9280         \fp_min:nn
9281         { \dim_ratio:nn \g_@@_tmpd_dim { \box_wd:N \l_tmpa_box } }
9282         { \dim_ratio:nn \g_tmpb_dim { \box_ht_plus_dp:N \l_tmpa_box } }
9283       }
9284       { 1.0 }
9285     }
9286     \box_scale:Nnn \l_tmpa_box { \fp_use:N \l_tmpa_fp } { \fp_use:N \l_tmpa_fp }
9287     \pgfpicture
9288     \pgfrememberpicturerepositiononpagetrue
9289     \pgf@relevantforpicturesizefalse
9290     \pgftransformshift
9291     {
9292       \pgfpoint
9293       { 0.5 * ( \g_@@_tmpc_dim + \g_@@_tmpe_dim ) }
9294       { \dim_use:N \g_tmpa_dim }
9295     }
9296     \pgfnode
9297     { rectangle }
9298     { center }
9299     { \box_use:N \l_tmpa_box }
9300     { }
9301     { }
9302     \endpgfpicture

```

```

9303     }
9304   }
9305 }

```

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.

```

9306 \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.

```

9307 \bool_new:N \g_@@_footnote_bool

9308 \msg_new:nnnn { nicematrix } { Unknown~key~for~package }
9309 {
9310   The~key~'\l_keys_key_str'~is~unknown. \\
9311   That~key~will~be~ignored. \\
9312   For~a~list~of~the~available~keys,~type~H~<return>.
9313 }
9314 {
9315   The~available~keys~are~(in~alphabetic~order):~
9316   footnote,~
9317   footnotehyper,~
9318   messages~for~Overleaf,~
9319   no~test~for~array,~
9320   renew~dots,~and~
9321   renew~matrix.
9322 }

9323 \keys_define:nn { nicematrix / Package }
9324 {
9325   renew~dots .bool_set:N = \l_@@_renew_dots_bool ,
9326   renew~dots .value_forbidden:n = true ,
9327   renew~matrix .code:n = \@@_renew_matrix: ,
9328   renew~matrix .value_forbidden:n = true ,
9329   messages~for~Overleaf .bool_set:N = \g_@@_messages_for_Overleaf_bool ,
9330   footnote .bool_set:N = \g_@@_footnote_bool ,
9331   footnotehyper .bool_set:N = \g_@@_footnotehyper_bool ,
9332   no~test~for~array .bool_set:N = \g_@@_no_test_for_array_bool ,
9333   no~test~for~array .default:n = true ,
9334   unknown .code:n = \@@_error:n { Unknown~key~for~package }
9335 }

9336 \ProcessKeysOptions { nicematrix / Package }

9337 \@@_msg_new:nn { footnote~with~footnotehyper~package }
9338 {
9339   You~can't~use~the~option~'footnote'~because~the~package~
9340   footnotehyper~has~already~been~loaded.~
9341   If~you~want,~you~can~use~the~option~'footnotehyper'~and~the~footnotes~
9342   within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
9343   of~the~package~footnotehyper.\\
9344   The~package~footnote~won't~be~loaded.
9345 }

```

```

9346 \@@_msg_new:nn { footnotehyper~with~footnote~package }
9347 {
9348   You~can't~use~the~option~'footnotehyper'~because~the~package~
9349   footnote~has~already~been~loaded.~
9350   If~you~want,~you~can~use~the~option~'footnote'~and~the~footnotes~
9351   within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
9352   of~the~package~footnote.\\
9353   The~package~footnotehyper~won't~be~loaded.
9354 }

```

```

9355 \bool_if:NT \g_@@_footnote_bool
9356 {

```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```

9357   \IfClassLoadedTF { beamer }
9358   { \bool_set_false:N \g_@@_footnote_bool }
9359   {
9360     \IfPackageLoadedTF { footnotehyper }
9361     { \@@_error:n { footnote~with~footnotehyper~package } }
9362     { \usepackage { footnote } }
9363   }
9364 }

```

```

9365 \bool_if:NT \g_@@_footnotehyper_bool
9366 {

```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```

9367   \IfClassLoadedTF { beamer }
9368   { \bool_set_false:N \g_@@_footnote_bool }
9369   {
9370     \IfPackageLoadedTF { footnote }
9371     { \@@_error:n { footnotehyper~with~footnote~package } }
9372     { \usepackage { footnotehyper } }
9373   }
9374   \bool_set_true:N \g_@@_footnote_bool
9375 }

```

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.

```

9376 \bool_new:N \l_@@_underscore_loaded_bool
9377 \IfPackageLoadedT { underscore }
9378 { \bool_set_true:N \l_@@_underscore_loaded_bool }
9379 \hook_gput_code:nnn { begindocument } { . }
9380 {
9381   \bool_if:NF \l_@@_underscore_loaded_bool
9382   {
9383     \IfPackageLoadedT { underscore }
9384     { \@@_error:n { underscore~after~nicematrix } }
9385   }
9386 }

```

40 Error messages of the package

```

9387 \bool_if:NTF \g_@@_messages_for_Overleaf_bool
9388 { \str_const:Nn \c_@@_available_keys_str { } }
9389 {
9390   \str_const:Nn \c_@@_available_keys_str
9391     { For~a~list~of~the~available~keys,~type~H~<return>. }
9392 }

9393 \seq_new:N \g_@@_types_of_matrix_seq
9394 \seq_gset_from_clist:Nn \g_@@_types_of_matrix_seq
9395 {
9396   NiceMatrix ,
9397   pNiceMatrix , bNiceMatrix , vNiceMatrix, BNiceMatrix, VNiceMatrix
9398 }
9399 \seq_gset_map_x:NNn \g_@@_types_of_matrix_seq \g_@@_types_of_matrix_seq
9400 { \tl_to_str:n { #1 } }

```

If the user uses too much columns, the command `\@@_error_too_much_cols:` is triggered. This command raises an error but also tries to give the best information to the user in the error message. The command `\seq_if_in:NoF` is not expandable and that's why we can't put it in the error message itself. We have to do the test before the `\@@_fatal:n`.

```

9401 \cs_new_protected:Npn \@@_error_too_much_cols:
9402 {
9403   \seq_if_in:NoF \g_@@_types_of_matrix_seq \g_@@_name_env_str
9404     { \@@_fatal:nn { too-much-cols-for-array } }
9405   \int_compare:nNnT \l_@@_last_col_int = { -2 }
9406     { \@@_fatal:n { too-much-cols-for-matrix } }
9407   \int_compare:nNnT \l_@@_last_col_int = { -1 }
9408     { \@@_fatal:n { too-much-cols-for-matrix } }
9409   \bool_if:NF \l_@@_last_col_without_value_bool
9410     { \@@_fatal:n { too-much-cols-for-matrix-with-last-col } }
9411 }

```

The following command must *not* be protected since it's used in an error message.

```

9412 \cs_new:Npn \@@_message_hdotsfor:
9413 {
9414   \tl_if_empty:oF \g_@@_HVdotsfor_lines_tl
9415     { ~Maybe~your~use~of~\token_to_str:N \Hdotsfor\ is~incorrect.}
9416 }

9417 \@@_msg_new:nn { hvlines,~rounded-corners~and~corners }
9418 {
9419   Incompatible~options.\\
9420   You~should~not~use~'hvlines',~'rounded-corners'~and~'corners'~at~this~time.\\
9421   The~output~will~not~be~reliable.
9422 }

9423 \@@_msg_new:nn { negative~weight }
9424 {
9425   Negative~weight.\\
9426   The~weight~of~the~'X'~columns~must~be~positive~and~you~have~used~
9427   the~value~'\int_use:N \l_@@_weight_int'.\\
9428   The~absolute~value~will~be~used.
9429 }

9430 \@@_msg_new:nn { last-col-not-used }
9431 {
9432   Column~not~used.\\
9433   The~key~'last-col'~is~in~force~but~you~have~not~used~that~last~column~
9434   in~your~\@@_full_name_env:.~However,~you~can~go~on.
9435 }

9436 \@@_msg_new:nn { too-much-cols-for-matrix-with-last-col }
9437 {

```

```

9438     Too-much-columns.\\
9439     In-the-row~\int_eval:n { \c@iRow },~
9440     you-try-to-use-more-columns~
9441     than-allowed-by-your~\@@_full_name_env:.\@@_message_hdotsfor:\
9442     The-maximal-number-of-columns-is~\int_eval:n { \l_@@_last_col_int - 1 }~
9443     (plus-the-exterior-columns).~This-error-is-fatal.
9444 }

9445 \@@_msg_new:nn { too-much-cols-for-matrix }
9446 {
9447     Too-much-columns.\\
9448     In-the-row~\int_eval:n { \c@iRow },~
9449     you-try-to-use-more-columns-than-allowed-by-your~
9450     \@@_full_name_env:.\@@_message_hdotsfor:\ Recall-that-the-maximal~
9451     number-of-columns-for-a-matrix~(excepted-the-potential-exterior~
9452     columns)-is-fixed-by-the-LaTeX-counter-'MaxMatrixCols'.~
9453     Its-current-value-is~\int_use:N \c@MaxMatrixCols\ (use~
9454     \token_to_str:N \setcounter\ to-change-that-value).~
9455     This-error-is-fatal.
9456 }

9457 \@@_msg_new:nn { too-much-cols-for-array }
9458 {
9459     Too-much-columns.\\
9460     In-the-row~\int_eval:n { \c@iRow },~
9461     ~you-try-to-use-more-columns-than-allowed-by-your~
9462     \@@_full_name_env:.\@@_message_hdotsfor:\ The-maximal-number-of-columns-is~
9463     \int_use:N \g_@@_static_num_of_col_int\
9464     ~(plus-the-potential-exterior-ones).~
9465     This-error-is-fatal.
9466 }

9467 \@@_msg_new:nn { columns-not-used }
9468 {
9469     Columns-not-used.\\
9470     The-preamble-of-your~\@@_full_name_env:\ announces~\int_use:N
9471     \g_@@_static_num_of_col_int\ columns-but-you-use-only~\int_use:N \c@jCol.\\
9472     The-columns-you-did-not-used-won't-be-created.\\
9473     You-won't-have-similar-error-message-till-the-end-of-the-document.
9474 }

9475 \@@_msg_new:nn { empty-preamble }
9476 {
9477     Empty-preamble.\\
9478     The-preamble-of-your~\@@_full_name_env:\ is-empty.\\
9479     This-error-is-fatal.
9480 }

9481 \@@_msg_new:nn { in-first-col }
9482 {
9483     Erroneous-use.\\
9484     You-can't-use-the-command-#1 in-the-first-column-(number-0)-of-the-array.\\
9485     That-command-will-be-ignored.
9486 }

9487 \@@_msg_new:nn { in-last-col }
9488 {
9489     Erroneous-use.\\
9490     You-can't-use-the-command-#1 in-the-last-column-(exterior)-of-the-array.\\
9491     That-command-will-be-ignored.
9492 }

9493 \@@_msg_new:nn { in-first-row }
9494 {
9495     Erroneous-use.\\
9496     You-can't-use-the-command-#1 in-the-first-row-(number-0)-of-the-array.\\
9497     That-command-will-be-ignored.

```

```

9498 }
9499 \@@_msg_new:nn { in~last~row }
9500 {
9501   You~can't~use~the~command~#1~in~the~last~row~(exterior)~of~the~array.\\
9502   That~command~will~be~ignored.
9503 }
9504 \@@_msg_new:nn { caption~outside~float }
9505 {
9506   Key~caption~forbidden.\\
9507   You~can't~use~the~key~'caption'~because~you~are~not~in~a~floating~
9508   environment.~This~key~will~be~ignored.
9509 }
9510 \@@_msg_new:nn { short~caption~without~caption }
9511 {
9512   You~should~not~use~the~key~'short~caption'~without~'caption'.~
9513   However,~your~'short~caption'~will~be~used~as~'caption'.
9514 }
9515 \@@_msg_new:nn { double~closing~delimiter }
9516 {
9517   Double~delimiter.\\
9518   You~can't~put~a~second~closing~delimiter~"#1"~just~after~a~first~closing~
9519   delimiter.~This~delimiter~will~be~ignored.
9520 }
9521 \@@_msg_new:nn { delimiter~after~opening }
9522 {
9523   Double~delimiter.\\
9524   You~can't~put~a~second~delimiter~"#1"~just~after~a~first~opening~
9525   delimiter.~That~delimiter~will~be~ignored.
9526 }
9527 \@@_msg_new:nn { bad~option~for~line~style }
9528 {
9529   Bad~line~style.\\
9530   Since~you~haven't~loaded~Tikz,~the~only~value~you~can~give~to~'line~style'~
9531   is~'standard'.~That~key~will~be~ignored.
9532 }
9533 \@@_msg_new:nn { Identical~notes~in~caption }
9534 {
9535   Identical~tabular~notes.\\
9536   You~can't~put~several~notes~with~the~same~content~in~
9537   \token_to_str:N \caption\ (but~you~can~in~the~main~tabular).\\
9538   If~you~go~on,~the~output~will~probably~be~erroneous.
9539 }
9540 \@@_msg_new:nn { tabularnote~below~the~tabular }
9541 {
9542   \token_to_str:N \tabularnote\ forbidden\\
9543   You~can't~use~\token_to_str:N \tabularnote\ in~the~caption~
9544   of~your~tabular~because~the~caption~will~be~composed~below~
9545   the~tabular.~If~you~want~the~caption~above~the~tabular~use~the~
9546   key~'caption~above'~in~\token_to_str:N \NiceMatrixOptions.\\
9547   Your~\token_to_str:N \tabularnote\ will~be~discarded~and~
9548   no~similar~error~will~raised~in~this~document.
9549 }
9550 \@@_msg_new:nn { Unknown~key~for~rules }
9551 {
9552   Unknown~key.\\
9553   There~is~only~two~keys~available~here:~width~and~color.\\
9554   Your~key~'\l_keys_key_str'~will~be~ignored.
9555 }
9556 \@@_msg_new:nn { Unknown~key~for~TikzEveryCell }

```

```

9557 {
9558   Unknown~key.\\
9559   There~is~only~two~keys~available~here::~
9560   'empty'~and~'not-empty'.\\
9561   Your~key~'\l_keys_key_str'~will~be~ignored.
9562 }
9563 \@@_msg_new:nn { Unknown~key~for~rotate }
9564 {
9565   Unknown~key.\\
9566   The~only~key~available~here~is~'c'.\\
9567   Your~key~'\l_keys_key_str'~will~be~ignored.
9568 }
9569 \@@_msg_new:nnn { Unknown~key~for~custom~line }
9570 {
9571   Unknown~key.\\
9572   The~key~'\l_keys_key_str'~is~unknown~in~a~'custom~line'.~
9573   It~you~go~on,~you~will~probably~have~other~errors. \\
9574   \c_@@_available_keys_str
9575 }
9576 {
9577   The~available~keys~are~(in~alphabetic~order):~
9578   ccommand,~
9579   color,~
9580   command,~
9581   dotted,~
9582   letter,~
9583   multiplicity,~
9584   sep-color,~
9585   tikz,~and~total-width.
9586 }
9587 \@@_msg_new:nnn { Unknown~key~for~xdots }
9588 {
9589   Unknown~key.\\
9590   The~key~'\l_keys_key_str'~is~unknown~for~a~command~for~drawing~dotted~rules.\\
9591   \c_@@_available_keys_str
9592 }
9593 {
9594   The~available~keys~are~(in~alphabetic~order):~
9595   'color',~
9596   'horizontal-labels',~
9597   'inter',~
9598   'line-style',~
9599   'radius',~
9600   'shorten',~
9601   'shorten-end'~and~'shorten-start'.
9602 }
9603 \@@_msg_new:nn { Unknown~key~for~rowcolors }
9604 {
9605   Unknown~key.\\
9606   As~for~now,~there~is~only~two~keys~available~here::~'cols'~and~'respect-blocks'~
9607   (and~you~try~to~use~'\l_keys_key_str')\\
9608   That~key~will~be~ignored.
9609 }
9610 \@@_msg_new:nn { label~without~caption }
9611 {
9612   You~can't~use~the~key~'label'~in~your~'{NiceTabular}'~because~
9613   you~have~not~used~the~key~'caption'.~The~key~'label'~will~be~ignored.
9614 }
9615 \@@_msg_new:nn { W~warning }
9616 {
9617   Line~\msg_line_number:.~The~cell~is~too~wide~for~your~column~'W'~

```



```

9618     (row~\int_use:N \c@iRow).
9619 }
9620 \@@_msg_new:nn { Construct-too-large }
9621 {
9622     Construct-too-large.\
9623     Your-command~\token_to_str:N #1
9624     can't-be-drawn-because-your-matrix-is-too-small.\
9625     That-command-will-be-ignored.
9626 }
9627 \@@_msg_new:nn { underscore-after-nicematrix }
9628 {
9629     Problem-with-'underscore'.\
9630     The-package~'underscore'~should-be-loaded-before~'nicematrix'.~
9631     You-can-go-on-but-you-won't-be-able-to-write-something-such-as:\
9632     '\token_to_str:N \Cdots\token_to_str:N _{n~\token_to_str:N \text{~times}}}'.
9633 }
9634 \@@_msg_new:nn { ampersand-in-light-syntax }
9635 {
9636     Ampersand-forbidden.\
9637     You-can't-use-an-ampersand~(\token_to_str:N &)~to-separate-columns-because~
9638     ~the-key~'light-syntax'~is-in-force.~This-error-is-fatal.
9639 }
9640 \@@_msg_new:nn { double-backslash-in-light-syntax }
9641 {
9642     Double-backslash-forbidden.\
9643     You-can't-use~\token_to_str:N
9644     \~to-separate-rows-because-the-key~'light-syntax'~
9645     is-in-force.~You-must-use-the-character~'\l_@@_end_of_row_tl'~
9646     (set-by-the-key~'end-of-row')~.~This-error-is-fatal.
9647 }
9648 \@@_msg_new:nn { hlines-with-color }
9649 {
9650     Incompatible-keys.\
9651     You-can't-use-the-keys~'hlines',~'vlines'~or~'hvlines'~for~a~
9652     '\token_to_str:N \Block'~when~the-key~'color'~or~'draw'~is-used.\
9653     However,~you-can-put~several-commands~\token_to_str:N \Block.\
9654     Your-key-will-be-discarded.
9655 }
9656 \@@_msg_new:nn { bad-value-for-baseline }
9657 {
9658     Bad-value-for-baseline.\
9659     The-value-given-to~'baseline'~(\int_use:N \l_tmpa_int)~is-not~
9660     valid.~The-value-must-be-between~\int_use:N \l_@@_first_row_int~and~
9661     \int_use:N \g_@@_row_total_int~or-equal-to~'t',~'c'~or~'b'~or-of~
9662     the-form~'line-i'~.\
9663     A-value-of~1~will-be-used.
9664 }
9665 \@@_msg_new:nn { detection-of-empty-cells }
9666 {
9667     Problem-with~'not-empty'~\
9668     For-technical-reasons,~you-must-activate~
9669     'create-cell-nodes'~in~\token_to_str:N \CodeBefore\
9670     in-order-to-use-the-key~'\l_keys_key_str'~.\
9671     That-key-will-be-ignored.
9672 }
9673 \@@_msg_new:nn { siunitx-not-loaded }
9674 {
9675     siunitx-not-loaded~\
9676     You-can't-use-the-columns~'S'~because~'siunitx'~is-not-loaded.\
9677     That-error-is-fatal.

```

```

9678     }
9679 \@@_msg_new:nn { ragged2e~not~loaded }
9680 {
9681     You-have-to-load~'ragged2e'~in-order-to-use-the-key~'\l_keys_key_str'~in~
9682     your~column~'\l_@@_vpos_col_str'~(or~'X').~The-key~'\str_lowercase:o
9683     \l_keys_key_str'~will-be-used~instead.
9684 }
9685 \@@_msg_new:nn { Invalid~name }
9686 {
9687     Invalid~name.\\
9688     You-can't-give-the-name~'\l_keys_value_tl'~to-a~\token_to_str:N
9689     \SubMatrix\ of-your~\@@_full_name_env:~\\
9690     A~name~must~be~accepted~by~the~regular~expression~[A-Za-z][A-Za-z0-9]*.\\
9691     This~key~will-be-ignored.
9692 }
9693 \@@_msg_new:nn { Wrong~line~in~SubMatrix }
9694 {
9695     Wrong~line.\\
9696     You-try-to-draw-a~#1~line-of-number~'#2'~in-a~
9697     \token_to_str:N \SubMatrix\ of-your~\@@_full_name_env:\ but-that~
9698     number~is-not-valid.~It~will-be-ignored.
9699 }
9700 \@@_msg_new:nn { Impossible~delimiter }
9701 {
9702     Impossible~delimiter.\\
9703     It's-impossible-to-draw-the~#1~delimiter-of-your~
9704     \token_to_str:N \SubMatrix\ because-all-the-cells-are-empty~
9705     in~that~column.
9706     \bool_if:NT \l_@@_submatrix_slim_bool
9707     { ~Maybe-you-should-try-without-the-key~'slim'. } \\
9708     This~\token_to_str:N \SubMatrix\ will-be-ignored.
9709 }
9710 \@@_msg_new:nnn { width~without~X~columns }
9711 {
9712     You-have-used-the-key~'width'~but-you-have-put-no~'X'~column.~
9713     That~key~will-be-ignored.
9714 }
9715 {
9716     This~message-is-the-message~'width~without~X~columns'~
9717     of~the~module~'nicematrix'.~
9718     The~experimented-users-can-disable~that~message-with~
9719     \token_to_str:N \msg_redirect_name:nnn.\\
9720 }
9721
9722 \@@_msg_new:nn { key~multiplicity~with~dotted }
9723 {
9724     Incompatible~keys. \\
9725     You-have-used-the-key~'multiplicity'~with-the-key~'dotted'~
9726     in-a~'custom-line'.~They~are~incompatible. \\
9727     The-key~'multiplicity'~will-be-discarded.
9728 }
9729 \@@_msg_new:nn { empty~environment }
9730 {
9731     Empty~environment.\\
9732     Your~\@@_full_name_env:\ is-empty.~This~error~is~fatal.
9733 }
9734 \@@_msg_new:nn { No~letter~and~no~command }
9735 {
9736     Erroneous~use.\\
9737     Your~use~of~'custom-line'~is~no-op~since~you~don't~have~used~the~

```

```

9738     key~'letter'~(for~a~letter~for~vertical~rules)~nor~the~keys~'command'~or~
9739     ~'ccommand'~(to~draw~horizontal~rules).\\
9740     However,~you~can~go~on.
9741 }
9742 \@@_msg_new:nn { Forbidden~letter }
9743 {
9744     Forbidden~letter.\\
9745     You~can't~use~the~letter~'#1'~for~a~customized~line.\\
9746     It~will~be~ignored.
9747 }
9748 \@@_msg_new:nn { Several~letters }
9749 {
9750     Wrong~name.\\
9751     You~must~use~only~one~letter~as~value~for~the~key~'letter'~(and~you~
9752     have~used~'\l_@@_letter_str').\\
9753     It~will~be~ignored.
9754 }
9755 \@@_msg_new:nn { Delimiter~with~small }
9756 {
9757     Delimiter~forbidden.\\
9758     You~can't~put~a~delimiter~in~the~preamble~of~your~\@@_full_name_env:\\
9759     because~the~key~'small'~is~in~force.\\
9760     This~error~is~fatal.
9761 }
9762 \@@_msg_new:nn { unknown~cell~for~line~in~CodeAfter }
9763 {
9764     Unknown~cell.\\
9765     Your~command~\token_to_str:N\line\{#1\}\{#2\}~in~
9766     the~\token_to_str:N \CodeAfter\ of~your~\@@_full_name_env:\\
9767     can't~be~executed~because~a~cell~doesn't~exist.\\
9768     This~command~\token_to_str:N \line\ will~be~ignored.
9769 }
9770 \@@_msg_new:nnn { Duplicate~name~for~SubMatrix }
9771 {
9772     Duplicate~name.\\
9773     The~name~'#1'~is~already~used~for~a~\token_to_str:N \SubMatrix\\
9774     in~this~\@@_full_name_env:.\\
9775     This~key~will~be~ignored.\\
9776     \bool_if:NF \g_@@_messages_for_Overleaf_bool
9777     { For~a~list~of~the~names~already~used,~type~H~<return>. }
9778 }
9779 {
9780     The~names~already~defined~in~this~\@@_full_name_env:\ are:~
9781     \seq_use:Nnnn \g_@@_submatrix_names_seq { ~and~ } { ,~ } { ~and~ }.
9782 }
9783 \@@_msg_new:nn { r~or~l~with~preamble }
9784 {
9785     Erroneous~use.\\
9786     You~can't~use~the~key~'\l_keys_key_str'~in~your~\@@_full_name_env:~
9787     You~must~specify~the~alignment~of~your~columns~with~the~preamble~of~
9788     your~\@@_full_name_env:.\\
9789     This~key~will~be~ignored.
9790 }
9791 \@@_msg_new:nn { Hdotsfor~in~col~0 }
9792 {
9793     Erroneous~use.\\
9794     You~can't~use~\token_to_str:N \Hdotsfor\ in~an~exterior~column~of~
9795     the~array.~This~error~is~fatal.
9796 }
9797 \@@_msg_new:nn { bad~corner }

```

```

9798 {
9799   Bad-corner.\
9800   #1-is-an-incorrect-specification-for-a-corner~(in-the-key~
9801   'corners').~The-available-values-are:~NW,~SW,~NE-and~SE.\
9802   This-specification-of~corner~will-be-ignored.
9803 }
9804 \@@_msg_new:nn { bad-border }
9805 {
9806   Bad-border.\
9807   \l_keys_key_str\space-is-an-incorrect-specification-for-a-border~
9808   (in-the-key~'borders'~of-the-command~\token_to_str:N \Block).~
9809   The-available-values-are:~left,~right,~top-and~bottom~(and-you-can~
9810   also-use-the-key~'tikz'
9811   \IfPackageLoadedF { tikz }
9812   {-if-you-load-the-LaTeX-package~'tikz'}).\\
9813   This-specification-of~border~will-be-ignored.
9814 }
9815 \@@_msg_new:nn { TikzEveryCell~without~tikz }
9816 {
9817   TikZ-not-loaded.\
9818   You-can't-use~\token_to_str:N \TikzEveryCell\
9819   because-you-have-not-loaded-tikz.~
9820   This-command-will-be-ignored.
9821 }
9822 \@@_msg_new:nn { tikz-key~without~tikz }
9823 {
9824   TikZ-not-loaded.\
9825   You-can't-use-the-key~'tikz'~for-the-command~'\token_to_str:N
9826   \Block'~because-you-have-not-loaded-tikz.~
9827   This-key-will-be-ignored.
9828 }
9829 \@@_msg_new:nn { last-col~non~empty~for~NiceArray }
9830 {
9831   Erroneous-use.\
9832   In-the~\@@_full_name_env:,~you-must-use-the-key~
9833   'last-col'~without-value.\
9834   However,~you-can-go-on-for~this-time~
9835   (the-value~'\l_keys_value_tl'~will-be-ignored).
9836 }
9837 \@@_msg_new:nn { last-col~non~empty~for~NiceMatrixOptions }
9838 {
9839   Erroneous-use.\
9840   In~\token_to_str:N \NiceMatrixOptions,~you-must-use-the-key~
9841   'last-col'~without-value.\
9842   However,~you-can-go-on-for~this-time~
9843   (the-value~'\l_keys_value_tl'~will-be-ignored).
9844 }
9845 \@@_msg_new:nn { Block~too~large~1 }
9846 {
9847   Block~too~large.\
9848   You-try-to-draw-a-block-in-the-cell~#1-#2-of~your-matrix-but~the-matrix-is~
9849   too-small~for~that-block. \
9850   This-block-and~maybe~others-will-be-ignored.
9851 }
9852 \@@_msg_new:nn { Block~too~large~2 }
9853 {
9854   Block~too~large.\
9855   The-preamble-of~your~\@@_full_name_env:\ announces~\int_use:N
9856   \g_@@_static_num_of_col_int\
9857   columns~but~you-use-only~\int_use:N \c@jCol\ and~that's~why~a~block~
9858   specified-in-the-cell~#1-#2~can't~be~drawn.~You-should~add~some~ampersands~

```

```

9859      (&)-at-the-end-of-the-first-row-of-your~\@@_full_name_env:.\
9860      This~block~and~maybe~others~will~be~ignored.
9861  }

9862  \@@_msg_new:nn { unknown~column-type }
9863  {
9864      Bad-column-type.\
9865      The-column-type~'#1'~in-your~\@@_full_name_env:\
9866      is~unknown. \
9867      This~error~is~fatal.
9868  }

9869  \@@_msg_new:nn { unknown~column-type-S }
9870  {
9871      Bad-column-type.\
9872      The-column-type~'S'~in-your~\@@_full_name_env:\ is~unknown. \
9873      If~you~want~to~use~the~column-type~'S'~of~siunitx,~you~should~
9874      load~that~package. \
9875      This~error~is~fatal.
9876  }

9877  \@@_msg_new:nn { tabularnote~forbidden }
9878  {
9879      Forbidden-command.\
9880      You~can't~use~the~command~\token_to_str:N\tabularnote\
9881      ~here.~This~command~is~available~only~in~
9882      \{NiceTabular\},~\{NiceTabular*\}~and~\{NiceTabularX\}~or~in~
9883      the~argument~of~a~command~\token_to_str:N \caption\ included~
9884      in~an~environment~{table}. \
9885      This~command~will~be~ignored.
9886  }

9887  \@@_msg_new:nn { borders~forbidden }
9888  {
9889      Forbidden-key.\
9890      You~can't~use~the~key~'borders'~of~the~command~\token_to_str:N \Block\
9891      because~the~option~'rounded-corners'~
9892      is~in~force~with~a~non-zero~value.\
9893      This~key~will~be~ignored.
9894  }

9895  \@@_msg_new:nn { bottomrule~without~booktabs }
9896  {
9897      booktabs~not~loaded.\
9898      You~can't~use~the~key~'tabular/bottomrule'~because~you~haven't~
9899      loaded~'booktabs'.\
9900      This~key~will~be~ignored.
9901  }

9902  \@@_msg_new:nn { enumitem~not~loaded }
9903  {
9904      enumitem~not~loaded.\
9905      You~can't~use~the~command~\token_to_str:N\tabularnote\
9906      ~because~you~haven't~loaded~'enumitem'.\
9907      All~the~commands~\token_to_str:N\tabularnote\ will~be~
9908      ignored~in~the~document.
9909  }

9910  \@@_msg_new:nn { tikz~without~tikz }
9911  {
9912      Tikz~not~loaded.\
9913      You~can't~use~the~key~'tikz'~here~because~Tikz~is~not~
9914      loaded.~If~you~go~on,~that~key~will~be~ignored.
9915  }

9916  \@@_msg_new:nn { tikz~in~custom~line~without~tikz }
9917  {
9918      Tikz~not~loaded.\

```

```

9919     You-have-used-the-key~'tikz'~in-the-definition-of-a~
9920     customized~line~(with~'custom-line')~but~tikz-is-not-loaded.~
9921     You-can-go-on-but-you-will-have-another-error-if-you-actually~
9922     use-that-custom-line.
9923 }
9924 \@@_msg_new:nn { tikz-in-borders-without-tikz }
9925 {
9926     Tikz-not-loaded.\\
9927     You-have-used-the-key~'tikz'~in-a-key~'borders'~(of-a~
9928     command~'\token_to_str:N\Block')~but~tikz-is-not-loaded.~
9929     That-key-will-be-ignored.
9930 }
9931 \@@_msg_new:nn { without-color-inside }
9932 {
9933     If-order-to-use~\token_to_str:N \cellcolor,~\token_to_str:N \rowcolor,~
9934     \token_to_str:N \rowcolors\ or~\token_to_str:N \rowlistcolors\
9935     outside~\token_to_str:N \CodeBefore,~you~
9936     should-have-used-the-key~'color-inside'~in-your~\@@_full_name_env:~\\
9937     You-can-go-on-but-you-may-need-more-compilations.
9938 }
9939 \@@_msg_new:nn { color-in-custom-line-with-tikz }
9940 {
9941     Erroneous-use.\\
9942     In-a~'custom-line',~you-have-used-both~'tikz'~and~'color',~
9943     which-is-forbidden~(you-should-use~'color'~inside~the-key~'tikz').~
9944     The-key~'color'~will-be-discarded.
9945 }
9946 \@@_msg_new:nn { Wrong-last-row }
9947 {
9948     Wrong-number.\\
9949     You-have-used~'last-row=~\int_use:N \l_@@_last_row_int'~but-your~
9950     \@@_full_name_env:\ seems-to-have~\int_use:N \c@iRow \ rows.~
9951     If-you-go-on,~the-value-of~\int_use:N \c@iRow \ will-be-used-for~
9952     last-row.~You-can-avoid-this-problem-by-using~'last-row'~
9953     without-value~(more-compilations-might-be-necessary).
9954 }
9955 \@@_msg_new:nn { Yet-in-env }
9956 {
9957     Nested-environments.\\
9958     Environments-of~nicematrix~can't-be-nested.\\
9959     This-error-is-fatal.
9960 }
9961 \@@_msg_new:nn { Outside-math-mode }
9962 {
9963     Outside-math-mode.\\
9964     The~\@@_full_name_env:\ can-be-used-only~in-math-mode~
9965     (and-not-in~\token_to_str:N \vcenter).\\
9966     This-error-is-fatal.
9967 }
9968 \@@_msg_new:nn { One-letter-allowed }
9969 {
9970     Bad-name.\\
9971     The-value-of-key~'\l_keys_key_str'~must-be-of-length~1.\\
9972     It-will-be-ignored.
9973 }
9974 \@@_msg_new:nn { TabularNote-in-CodeAfter }
9975 {
9976     Environment~{TabularNote}~forbidden.\\
9977     You-must-use~{TabularNote}~at-the-end-of-your~{NiceTabular}~
9978     but~*before*~the~\token_to_str:N \CodeAfter.\\

```

```

9979     This~environment~{TabularNote}~will~be~ignored.
9980 }
9981 \@@_msg_new:nn { varwidth~not~loaded }
9982 {
9983     varwidth~not~loaded.\\
9984     You~can't~use~the~column~type~'V'~because~'varwidth'~is~not~
9985     loaded.\\
9986     Your~column~will~behave~like~'p'.
9987 }
9988 \@@_msg_new:nnn { Unknow~key~for~RulesBis }
9989 {
9990     Unknow~key.\\
9991     Your~key~'\l_keys_key_str'~is~unknown~for~a~rule.\\
9992     \c_@@_available_keys_str
9993 }
9994 {
9995     The~available~keys~are~(in~alphabetic~order):~
9996     color,~
9997     dotted,~
9998     multiplicity,~
9999     sep~color,~
10000     tikz,~and~total~width.
10001 }
10002
10003 \@@_msg_new:nnn { Unknown~key~for~Block }
10004 {
10005     Unknown~key.\\
10006     The~key~'\l_keys_key_str'~is~unknown~for~the~command~\token_to_str:N
10007     \Block.\\ It~will~be~ignored. \\
10008     \c_@@_available_keys_str
10009 }
10010 {
10011     The~available~keys~are~(in~alphabetic~order):~&~in~blocks,~ampersand~in~blocks,~
10012     b,~B,~borders,~c,~draw,~fill,~hlines,~hvlines,~l,~line~width,~name,~
10013     opacity,~rounded~corners,~r,~respect~arraystretch,~t,~T,~tikz,~transparent~
10014     and~vlines.
10015 }
10016 \@@_msg_new:nnn { Unknown~key~for~Brace }
10017 {
10018     Unknown~key.\\
10019     The~key~'\l_keys_key_str'~is~unknown~for~the~commands~\token_to_str:N
10020     \UnderBrace\ and~\token_to_str:N \OverBrace.\\
10021     It~will~be~ignored. \\
10022     \c_@@_available_keys_str
10023 }
10024 {
10025     The~available~keys~are~(in~alphabetic~order):~color,~left~shorten,~
10026     right~shorten,~shorten~(which~fixes~both~left~shorten~and~
10027     right~shorten)~and~yshift.
10028 }
10029 \@@_msg_new:nnn { Unknown~key~for~CodeAfter }
10030 {
10031     Unknown~key.\\
10032     The~key~'\l_keys_key_str'~is~unknown.\\
10033     It~will~be~ignored. \\
10034     \c_@@_available_keys_str
10035 }
10036 {
10037     The~available~keys~are~(in~alphabetic~order):~
10038     delimiters/color,~
10039     rules~(with~the~subkeys~'color'~and~'width'),~
10040     sub~matrix~(several~subkeys)~

```

```

10041     and~xdots~(several~subkeys).~
10042     The~latter~is~for~the~command~\token_to_str:N \line.
10043 }

10044 \@@_msg_new:nnn { Unknown~key~for~CodeBefore }
10045 {
10046     Unknown~key.\\
10047     The~key~'\l_keys_key_str'~is~unknown.\\
10048     It~will~be~ignored. \\
10049     \c_@@_available_keys_str
10050 }
10051 {
10052     The~available~keys~are~(in~alphabetic~order):~
10053     create~cell~nodes,~
10054     delimiters/color~and~
10055     sub~matrix~(several~subkeys).
10056 }

10057 \@@_msg_new:nnn { Unknown~key~for~SubMatrix }
10058 {
10059     Unknown~key.\\
10060     The~key~'\l_keys_key_str'~is~unknown.\\
10061     That~key~will~be~ignored. \\
10062     \c_@@_available_keys_str
10063 }
10064 {
10065     The~available~keys~are~(in~alphabetic~order):~
10066     'delimiters/color',~
10067     'extra~height',~
10068     'hlines',~
10069     'hvlines',~
10070     'left~xshift',~
10071     'name',~
10072     'right~xshift',~
10073     'rules'~(with~the~subkeys~'color'~and~'width'),~
10074     'slim',~
10075     'vlines'~and~'xshift'~(which~sets~both~'left~xshift'~
10076     and~'right~xshift').\\
10077 }

10078 \@@_msg_new:nnn { Unknown~key~for~notes }
10079 {
10080     Unknown~key.\\
10081     The~key~'\l_keys_key_str'~is~unknown.\\
10082     That~key~will~be~ignored. \\
10083     \c_@@_available_keys_str
10084 }
10085 {
10086     The~available~keys~are~(in~alphabetic~order):~
10087     bottomrule,~
10088     code~after,~
10089     code~before,~
10090     detect~duplicates,~
10091     enumitem~keys,~
10092     enumitem~keys~para,~
10093     para,~
10094     label~in~list,~
10095     label~in~tabular~and~
10096     style.
10097 }

10098 \@@_msg_new:nnn { Unknown~key~for~RowStyle }
10099 {
10100     Unknown~key.\\
10101     The~key~'\l_keys_key_str'~is~unknown~for~the~command~
10102     \token_to_str:N \RowStyle. \\

```



```

10103     That~key~will~be~ignored. \\
10104     \c_@@_available_keys_str
10105 }
10106 {
10107     The~available~keys~are~(in~alphabetic~order):~
10108     'bold',~
10109     'cell-space-top-limit',~
10110     'cell-space-bottom-limit',~
10111     'cell-space-limits',~
10112     'color',~
10113     'nb-rows'~and~
10114     'rowcolor'.
10115 }
10116 \@@_msg_new:nmn { Unknown~key~for~NiceMatrixOptions }
10117 {
10118     Unknown~key.\\
10119     The~key~'\l_keys_key_str'~is~unknown~for~the~command~
10120     \token_to_str:N \NiceMatrixOptions. \\
10121     That~key~will~be~ignored. \\
10122     \c_@@_available_keys_str
10123 }
10124 {
10125     The~available~keys~are~(in~alphabetic~order):~
10126     &~in~blocks,~
10127     allow~duplicate~names,~
10128     ampersand~in~blocks,~
10129     caption~above,~
10130     cell-space-bottom-limit,~
10131     cell-space-limits,~
10132     cell-space-top-limit,~
10133     code~for~first~col,~
10134     code~for~first~row,~
10135     code~for~last~col,~
10136     code~for~last~row,~
10137     corners,~
10138     custom~key,~
10139     create~extra~nodes,~
10140     create~medium~nodes,~
10141     create~large~nodes,~
10142     custom~line,~
10143     delimiters~(several~subkeys),~
10144     end~of~row,~
10145     first~col,~
10146     first~row,~
10147     hlines,~
10148     hvlines,~
10149     hvlines~except~borders,~
10150     last~col,~
10151     last~row,~
10152     left~margin,~
10153     light~syntax,~
10154     light~syntax~expanded,~
10155     matrix/columns~type,~
10156     no~cell~nodes,~
10157     notes~(several~subkeys),~
10158     nullify~dots,~
10159     pgf~node~code,~
10160     renew~dots,~
10161     renew~matrix,~
10162     respect~arraystretch,~
10163     rounded~corners,~
10164     right~margin,~
10165     rules~(with~the~subkeys~'color'~and~'width'),~

```

```

10166     small,~
10167     sub-matrix~(several~subkeys),~
10168     vlines,~
10169     xdots~(several~subkeys).
10170 }

```

For '{NiceArray}', the set of keys is the same as for {NiceMatrix} excepted that there is no 1 and r.

```

10171 \@@_msg_new:nnn { Unknown~key~for~NiceArray }
10172 {
10173   Unknown~key.\\
10174   The~key~'\l_keys_key_str'~is~unknown~for~the~environment~
10175   \{NiceArray\}. \\
10176   That~key~will~be~ignored. \\
10177   \c_@@_available_keys_str
10178 }
10179 {
10180   The~available~keys~are~(in~alphabetic~order):~
10181   &-in-blocks,~
10182   ampersand-in-blocks,~
10183   b,~
10184   baseline,~
10185   c,~
10186   cell-space-bottom-limit,~
10187   cell-space-limits,~
10188   cell-space-top-limit,~
10189   code-after,~
10190   code-for-first-col,~
10191   code-for-first-row,~
10192   code-for-last-col,~
10193   code-for-last-row,~
10194   color-inside,~
10195   columns-width,~
10196   corners,~
10197   create-extra-nodes,~
10198   create-medium-nodes,~
10199   create-large-nodes,~
10200   extra-left-margin,~
10201   extra-right-margin,~
10202   first-col,~
10203   first-row,~
10204   hlines,~
10205   hvlines,~
10206   hvlines-except-borders,~
10207   last-col,~
10208   last-row,~
10209   left-margin,~
10210   light-syntax,~
10211   light-syntax-expanded,~
10212   name,~
10213   no-cell-nodes,~
10214   nullify-dots,~
10215   pgf-node-code,~
10216   renew-dots,~
10217   respect-arraystretch,~
10218   right-margin,~
10219   rounded-corners,~
10220   rules~(with~the~subkeys~'color'~and~'width'),~
10221   small,~
10222   t,~
10223   vlines,~
10224   xdots/color,~
10225   xdots/shorten-start,~
10226   xdots/shorten-end,~

```

```

10227     xdots/shorten-and-
10228     xdots/line-style.
10229 }

```

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`).

```

10230 \@@_msg_new:nnn { Unknown~key~for~NiceMatrix }
10231 {
10232   Unknown~key.\\
10233   The~key~'\l_keys_key_str'~is~unknown~for~the~
10234   \@@_full_name_env:. \\
10235   That~key~will~be~ignored. \\
10236   \c_@@_available_keys_str
10237 }
10238 {
10239   The~available~keys~are~(in~alphabetic~order):~
10240   &-in-blocks,~
10241   ampersand-in-blocks,~
10242   b,~
10243   baseline,~
10244   c,~
10245   cell-space-bottom-limit,~
10246   cell-space-limits,~
10247   cell-space-top-limit,~
10248   code-after,~
10249   code-for-first-col,~
10250   code-for-first-row,~
10251   code-for-last-col,~
10252   code-for-last-row,~
10253   color-inside,~
10254   columns-type,~
10255   columns-width,~
10256   corners,~
10257   create-extra-nodes,~
10258   create-medium-nodes,~
10259   create-large-nodes,~
10260   extra-left-margin,~
10261   extra-right-margin,~
10262   first-col,~
10263   first-row,~
10264   hlines,~
10265   hvlines,~
10266   hvlines-except-borders,~
10267   l,~
10268   last-col,~
10269   last-row,~
10270   left-margin,~
10271   light-syntax,~
10272   light-syntax-expanded,~
10273   name,~
10274   no-cell-nodes,~
10275   nullify-dots,~
10276   pgf-node-code,~
10277   r,~
10278   renew-dots,~
10279   respect-arraystretch,~
10280   right-margin,~
10281   rounded-corners,~
10282   rules~(with~the~subkeys~'color'~and~'width'),~
10283   small,~
10284   t,~
10285   vlides,~
10286   xdots/color,~
10287   xdots/shorten-start,~

```

```

10288     xdots/shorten-end,~
10289     xdots/shorten-and~
10290     xdots/line-style.
10291 }
10292 \@@_msg_new:nnn { Unknown~key~for~NiceTabular }
10293 {
10294     Unknown~key.\\
10295     The~key~'\l_keys_key_str'~is~unknown~for~the~environment~
10296     \{NiceTabular\}. \\
10297     That~key~will~be~ignored. \\
10298     \c_@@_available_keys_str
10299 }
10300 {
10301     The~available~keys~are~(in~alphabetic~order):~
10302     &-in-blocks,~
10303     ampersand-in-blocks,~
10304     b,~
10305     baseline,~
10306     c,~
10307     caption,~
10308     cell-space-bottom-limit,~
10309     cell-space-limits,~
10310     cell-space-top-limit,~
10311     code-after,~
10312     code-for-first-col,~
10313     code-for-first-row,~
10314     code-for-last-col,~
10315     code-for-last-row,~
10316     color-inside,~
10317     columns-width,~
10318     corners,~
10319     custom-line,~
10320     create-extra-nodes,~
10321     create-medium-nodes,~
10322     create-large-nodes,~
10323     extra-left-margin,~
10324     extra-right-margin,~
10325     first-col,~
10326     first-row,~
10327     hlines,~
10328     hvlines,~
10329     hvlines-except-borders,~
10330     label,~
10331     last-col,~
10332     last-row,~
10333     left-margin,~
10334     light-syntax,~
10335     light-syntax-expanded,~
10336     name,~
10337     no-cell-nodes,~
10338     notes~(several~subkeys),~
10339     nullify-dots,~
10340     pgf-node-code,~
10341     renew-dots,~
10342     respect-arraystretch,~
10343     right-margin,~
10344     rounded-corners,~
10345     rules~(with~the~subkeys~'color'~and~'width'),~
10346     short-caption,~
10347     t,~
10348     tabularnote,~
10349     vlides,~
10350     xdots/color,~

```

```

10351     xdots/shorten-start,~
10352     xdots/shorten-end,~
10353     xdots/shorten-and~
10354     xdots/line-style.
10355 }
10356 \@@_msg_new:nnn { Duplicate-name }
10357 {
10358     Duplicate-name.\
10359     The-name-'\l_keys_value_tl'-is-already-used-and-you-shouldn't-use~
10360     the-same-environment-name-twice.~You-can-go-on,~but,~
10361     maybe,~you-will-have-incorrect-results-especially~
10362     if-you-use~'columns-width=auto'.~If-you-don't-want-to-see-this~
10363     message-again,~use-the-key~'allow-duplicate-names'-in~
10364     '\token_to_str:N \NiceMatrixOptions'.\
10365     \bool_if:NF \g_@@_messages_for_Overleaf_bool
10366     { For-a-list-of-the-names-already-used,~type-H~<return>. }
10367 }
10368 {
10369     The-names-already-defined-in-this-document-are:~
10370     \seq_use:Nnnn \g_@@_names_seq { ~and~ } { ,~ } { ~and~ }.
10371 }
10372 \@@_msg_new:nn { Option-auto-for-columns-width }
10373 {
10374     Erroneous-use.\
10375     You-can't-give-the-value~'auto'~to-the-key~'columns-width'~here.~
10376     That-key-will-be-ignored.
10377 }
10378 \@@_msg_new:nn { NiceTabularX~without~X }
10379 {
10380     NiceTabularX-without-X.\
10381     You-should-not-use~{NiceTabularX}~without~X-columns.\
10382     However,~you-can-go-on.
10383 }
10384 \@@_msg_new:nn { Preamble-forgotten }
10385 {
10386     Preamble-forgotten.\
10387     You-have-probably-forgotten-the-preamble-of-your~
10388     \@@_full_name_env:. \
10389     This-error-is-fatal.
10390 }

```

Contents

1	Declaration of the package and packages loaded	1
2	Security test	3
3	Collecting options	4
4	Technical definitions	5
5	Parameters	9
6	The command <code>\tabularnote</code>	20
7	Command for creation of rectangle nodes	24
8	The options	25
9	Important code used by <code>{NiceArrayWithDelims}</code>	36
10	The <code>\CodeBefore</code>	50
11	The environment <code>{NiceArrayWithDelims}</code>	53
12	We construct the preamble of the array	58
13	The redefinition of <code>\multicolumn</code>	74
14	The environment <code>{NiceMatrix}</code> and its variants	92
15	<code>{NiceTabular}</code> , <code>{NiceTabularX}</code> and <code>{NiceTabular*}</code>	93
16	After the construction of the array	94
17	We draw the dotted lines	101
18	The actual instructions for drawing the dotted lines with Tikz	114
19	User commands available in the new environments	120
20	The command <code>\line</code> accessible in code-after	126
21	The command <code>\RowStyle</code>	127
22	Colors of cells, rows and columns	130
23	The vertical and horizontal rules	142
24	The empty corners	157
25	The environment <code>{NiceMatrixBlock}</code>	160
26	The extra nodes	161
27	The blocks	165
28	How to draw the dotted lines transparently	189
29	Automatic arrays	189
30	The redefinition of the command <code>\dotfill</code>	191

31	The command <code>\diagbox</code>	191
32	The keyword <code>\CodeAfter</code>	192
33	The delimiters in the preamble	193
34	The command <code>\SubMatrix</code>	194
35	Les commandes <code>\UnderBrace</code> et <code>\OverBrace</code>	203
36	The command <code>TikzEveryCell</code>	206
37	The command <code>\ShowCellNames</code>	207
38	We process the options at package loading	210
39	About the package underscore	211
40	Error messages of the package	212