

The code of the package `nicematrix`*

F. Pantigny
fpantigny@wanadoo.fr

February 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`).

By default, the package `nicematrix` doesn't patch any existing code.

However, when the option `renew-dots` is used, the commands `\cdots`, `\ldots`, `\dots`, `\vdots`, `\ddots` and `\iddots` are redefined in the environments provided by `nicematrix`. In the same way, if the option `renew-matrix` is used, the environment `{matrix}` of `amsmath` is redefined.

On the other hand, the environment `{array}` is never redefined.

Of course, the package `nicematrix` uses the features of the package `array`. It tries to be independent of its implementation. Unfortunately, it was not possible to be strictly independent. For example, the package `nicematrix` relies upon the fact that the package `{array}` uses `\ialign` to begin the `\halign`.

1 Declaration of the package and packages loaded

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

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

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

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

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

```
3 \RequirePackage{l3keys2e}
4 \ProvidesExplPackage
5   {nicematrix}
6   {\myfiledate}
7   {\myfileversion}
8   {Enhanced arrays with the help of PGF/TikZ}
```

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

We load some packages.

```
9 \RequirePackage { array }
10 \RequirePackage { amsmath }
```

*This document corresponds to the version 6.27 of `nicematrix`, at the date of 2024/02/13.

```

11 \cs_new_protected:Npn \@@_error:n { \msg_error:nn { nicematrix } }
12 \cs_new_protected:Npn \@@_warning:n { \msg_warning:nn { nicematrix } }
13 \cs_new_protected:Npn \@@_error:nn { \msg_error:nnn { nicematrix } }
14 \cs_generate_variant:Nn \@@_error:nn { n e }
15 \cs_new_protected:Npn \@@_error:nnn { \msg_error:nnnn { nicematrix } }
16 \cs_new_protected:Npn \@@_fatal:n { \msg_fatal:nn { nicematrix } }
17 \cs_new_protected:Npn \@@_fatal:nn { \msg_fatal:nnn { nicematrix } }
18 \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).

```

19 \cs_new_protected:Npn \@@_msg_new:nnn #1 #2 #3
20 {
21   \bool_if:NTF \g_@@_messages_for_Overleaf_bool
22     { \msg_new:nnn { nicematrix } { #1 } { #2 \\\ #3 } }
23     { \msg_new:nnnn { nicematrix } { #1 } { #2 } { #3 } }
24 }

```

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

```

25 \cs_new_protected:Npn \@@_error_or_warning:n
26 { \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”.

```

27 \bool_new:N \g_@@_messages_for_Overleaf_bool
28 \bool_gset:Nn \g_@@_messages_for_Overleaf_bool
29 {
30   \str_if_eq_p:on \c_sys_jobname_str { _region_ } % for Emacs
31   || \str_if_eq_p:on \c_sys_jobname_str { output } % for Overleaf
32 }

```

```

33 \cs_new_protected:Npn \@@_msg_redirect_name:nn
34 { \msg_redirect_name:nnn { nicematrix } }
35 \cs_new_protected:Npn \@@_gredirect_none:n #1
36 {
37   \group_begin:
38   \globaldefs = 1
39   \@@_msg_redirect_name:nn { #1 } { none }
40   \group_end:
41 }
42 \cs_new_protected:Npn \@@_err_gredirect_none:n #1
43 {
44   \@@_error:n { #1 }
45   \@@_gredirect_none:n { #1 }
46 }
47 \cs_new_protected:Npn \@@_warning_gredirect_none:n #1
48 {
49   \@@_warning:n { #1 }
50   \@@_gredirect_none:n { #1 }
51 }

```

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.

```

52 \@@_msg_new:nn { Internal~error }
53 {
54   Potential~problem~when~using~nicematrix.\\
55   The~package~nicematrix~have~detected~a~modification~of~the~
56   standard~environment~{array}~(of~the~package~array).~Maybe~you~will~encounter~
57   some~slight~problems~when~using~nicematrix.~If~you~don't~want~to~see~
58   this~message~again,~load~nicematrix~with:~\token_to_str:N
59   \usepackage[no-test-for-array]{nicematrix}.
60 }

61 \@@_msg_new:nn { mdwtab~loaded }
62 {
63   The~packages~'mdwtab'~and~'nicematrix'~are~incompatible.~
64   This~error~is~fatal.
65 }

66 \cs_new_protected:Npn \@@_security_test:n #1
67 {
68   \peek_meaning:NTF \ignorespaces
69   { \@@_security_test_i:w }
70   { \@@_error:n { Internal~error } }
71   #1
72 }

73 \cs_new_protected:Npn \@@_security_test_i:w \ignorespaces #1
74 {
75   \peek_meaning:NF \unskip { \@@_error:n { Internal~error } }
76   #1
77 }

```

Here, the box `\l_tmpa_box` will be used as sandbox to take our security test. This code has been modified in version 6.18 (see question 682891 on TeX StackExchange).

```

78 \hook_gput_code:nnn { begindocument / after } { . }
79 {
80   \IfPackageLoadedTF { mdwtab }
81   { \@@_fatal:n { mdwtab~loaded } }
82   {
83     \bool_if:NF \g_@@_no_test_for_array_bool
84     {
85       \group_begin:
86       \hbox_set:Nn \l_tmpa_box
87       {
88         \begin { tabular } { c > { \@@_security_test:n } c c }
89         text & & text
90         \end { tabular }
91       }
92       \group_end:
93     }
94   }
95 }

```

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`).

```

96 \cs_new_protected:Npn \@@_collect_options:n #1
97 {
98   \peek_meaning:NTF [
99     { \@@_collect_options:nw { #1 } }
100     { #1 { } }
101   }

```

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

```

102 \NewDocumentCommand \@@_collect_options:nw { m r[] }
103 { \@@_collect_options:nn { #1 } { #2 } }
104
105 \cs_new_protected:Npn \@@_collect_options:nn #1 #2
106 {
107   \peek_meaning:NTF [
108     { \@@_collect_options:nnw { #1 } { #2 } }
109     { #1 { #2 } }
110   }
111
112 \cs_new_protected:Npn \@@_collect_options:nnw #1#2[#3]
113 { \@@_collect_options:nn { #1 } { #2 , #3 } }

```

4 Technical definitions

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

```

114 \tl_const:Nn \c_@@_b_tl { b }
115 \tl_const:Nn \c_@@_c_tl { c }
116 \tl_const:Nn \c_@@_l_tl { l }
117 \tl_const:Nn \c_@@_r_tl { r }
118 \tl_const:Nn \c_@@_all_tl { all }
119 \tl_const:Nn \c_@@_dot_tl { . }
120 \tl_const:Nn \c_@@_default_tl { default }
121 \tl_const:Nn \c_@@_star_tl { * }
122 \str_const:Nn \c_@@_r_str { r }
123 \str_const:Nn \c_@@_c_str { c }
124 \str_const:Nn \c_@@_l_str { l }
125 \str_const:Nn \c_@@_R_str { R }
126 \str_const:Nn \c_@@_C_str { C }
127 \str_const:Nn \c_@@_L_str { L }
128 \str_const:Nn \c_@@_j_str { j }
129 \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).

```

130 \tl_new:N \l_@@_argspec_tl
131 \cs_generate_variant:Nn \seq_set_split:Nnn { N V n }
132 \cs_generate_variant:Nn \str_lowercase:n { V }

133 \hook_gput_code:nnn { begindocument } { . }
134 {
135   \IfPackageLoadedTF { tikz }
136   {

```

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 `\AtBeginDocument` the correct version of some commands. The tokens `\exp_not:N` are mandatory.

```

137   \tl_const:Nn \c_@@_pgfortikzpicture_tl { \exp_not:N \tikzpicture }
138   \tl_const:Nn \c_@@_endpgfortikzpicture_tl { \exp_not:N \endtikzpicture }
139 }
140 {
141   \tl_const:Nn \c_@@_pgfortikzpicture_tl { \exp_not:N \pgfpicture }
142   \tl_const:Nn \c_@@_endpgfortikzpicture_tl { \exp_not:N \endpgfpicture }
143 }
144 }

```

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 May 2023, the current version `revtex4-2` is 4.2f (compatible with `booktabs`).

```

145 \IfClassLoadedTF { revtex4-1 }
146 { \bool_const:Nn \c_@@_revtex_bool \c_true_bool }
147 {
148   \IfClassLoadedTF { revtex4-2 }
149   { \bool_const:Nn \c_@@_revtex_bool \c_true_bool }
150   {

```

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

```

151   \cs_if_exist:NT \rvtx@ifformat@geq
152   { \bool_const:Nn \c_@@_revtex_bool \c_true_bool }
153   { \bool_const:Nn \c_@@_revtex_bool \c_false_bool }
154 }
155 }

```

```

156 \cs_generate_variant:Nn \tl_if_single_token_p:n { V }

```

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.

```

157 \cs_new_protected:Npn \@@_provide_pgfsyspdfmark:
158 {
159   \iow_now:Nn \@mainaux
160   {
161     \ExplSyntaxOn
162     \cs_if_free:NT \pgfsyspdfmark
163     { \cs_set_eq:NN \pgfsyspdfmark \@gobblethree }
164     \ExplSyntaxOff

```

```

165     }
166     \cs_gset_eq:NN \@@_provide_pgfsyspdfmark: \prg_do_nothing:
167 }

```

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.

```

168 \ProvideDocumentCommand \iddots { }
169 {
170   \mathinner
171   {
172     \tex_mkern:D 1 mu
173     \box_move_up:nn { 1 pt } { \hbox { . } }
174     \tex_mkern:D 2 mu
175     \box_move_up:nn { 4 pt } { \hbox { . } }
176     \tex_mkern:D 2 mu
177     \box_move_up:nn { 7 pt }
178     { \vbox:n { \kern 7 pt \hbox { . } } }
179     \tex_mkern:D 1 mu
180   }
181 }

```

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.

```

182 \hook_gput_code:nnn { begindocument } { . }
183 {
184   \IfPackageLoadedTF { booktabs }
185   { \iow_now:Nn \@mainaux \nicematrix@redefine@check@rerun }
186   { }
187 }
188 \cs_set_protected:Npn \nicematrix@redefine@check@rerun
189 {
190   \cs_set_eq:NN \@@_old_pgfutil@check@rerun \pgfutil@check@rerun

```

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

```

191   \cs_set_protected:Npn \pgfutil@check@rerun ##1 ##2
192   {
193     \str_if_eq:eeF { nm- } { \tl_range:nnn { ##1 } 1 3 }
194     { \@@_old_pgfutil@check@rerun { ##1 } { ##2 } }
195   }
196 }

```

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

```

197 \hook_gput_code:nnn { begindocument } { . }
198 {
199   \IfPackageLoadedTF { colortbl }
200   { }
201   {

```

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.

```

202     \cs_set_protected:Npn \CT@arc@ { }
203     \cs_set:Npn \arrayrulecolor #1 # { \CT@arc@ { #1 } }
204     \cs_set:Npn \CT@arc@ #1 #2
205     {
206       \dim_compare:nNnT \baselineskip = \c_zero_dim \noalign
207       { \cs_gset_nopar:Npn \CT@arc@ { \color #1 { #2 } } }
208     }

```

Idem for \CT@drs@.

```

209     \cs_set:Npn \doublerulesepcolor #1 # { \CT@drs { #1 } }
210     \cs_set:Npn \CT@drs #1 #2
211     {
212         \dim_compare:nNnT \baselineskip = \c_zero_dim \noalign
213         { \cs_gset:Npn \CT@drsc@ { \color #1 { #2 } } }
214     }
215     \cs_set:Npn \hline
216     {
217         \noalign { \ifnum 0 = ' } \fi
218         \cs_set_eq:NN \hskip \vskip
219         \cs_set_eq:NN \vrule \hrule
220         \cs_set_eq:NN \@width \@height
221         { \CT@arc@ \vline }
222         \futurelet \reserved@a
223         \@xhline
224     }
225 }
226 }
```

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.

```

227 \cs_set:Npn \@@_standard_cline #1 { \@@_standard_cline:w #1 \q_stop }
228 \cs_set:Npn \@@_standard_cline:w #1-#2 \q_stop
229 {
230     \int_if_zero:nT \l_@@_first_col_int { \omit & }
231     \int_compare:nNnT { #1 } > \c_one_int
232     { \multispan { \int_eval:n { #1 - 1 } } & }
233     \multispan { \int_eval:n { #2 - #1 + 1 } }
234     {
235         \CT@arc@
236         \leaders \hrule \@height \arrayrulewidth \hfill

```

The following \skip_horizontal:N \c_zero_dim is to prevent a potential \unskip to delete the \leaders¹

```

237     \skip_horizontal:N \c_zero_dim
238 }
```

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.

```

239     \everycr { }
240     \cr
241     \noalign { \skip_vertical:N -\arrayrulewidth }
242 }
```

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.

```

243 \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.

```

244 { \@@_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*.

```

245 \cs_set:Npn \@@_cline_i:nn #1 #2 { \@@_cline_i:w #1|#2- \q_stop }
246 \cs_set:Npn \@@_cline_i:w #1|#2-#3 \q_stop
247 {
```

¹See question 99041 on TeX StackExchange.

```

248 \tl_if_empty:nTF { #3 }
249 { \@@_cline_iii:w #1|#2-#2 \q_stop }
250 { \@@_cline_ii:w #1|#2-#3 \q_stop }
251 }
252 \cs_set:Npn \@@_cline_ii:w #1|#2-#3-\q_stop
253 { \@@_cline_iii:w #1|#2-#3 \q_stop }
254 \cs_set:Npn \@@_cline_iii:w #1|#2-#3 \q_stop
255 {

```

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).

```

256 \int_compare:nNt { #1 } < { #2 }
257 { \multispan { \int_eval:n { #2 - #1 } } & }
258 \multispan { \int_eval:n { #3 - #2 + 1 } }
259 {
260 \CT@arc@
261 \leaders \hrule \@height \arrayrulewidth \hfill
262 \skip_horizontal:N \c_zero_dim
263 }

```

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

```

264 \peek_meaning_remove_ignore_spaces:NTF \cline
265 { & \@@_cline_i:en { \int_eval:n { #3 + 1 } } }
266 { \everycr { } \cr }
267 }
268 \cs_generate_variant:Nn \@@_cline_i:nn { e n }

```

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

```

269 \cs_set_eq:NN \@@_math_toggle: \c_math_toggle_token

```

```

270 \cs_new_protected:Npn \@@_set_CT@arc@:n #1
271 {
272 \tl_if_blank:nF { #1 }
273 {
274 \tl_if_head_eq_meaning:nNTF { #1 } [
275 { \cs_set:Npn \CT@arc@ { \color #1 } }
276 { \cs_set:Npn \CT@arc@ { \color { #1 } } }
277 ]
278 }
279 \cs_generate_variant:Nn \@@_set_CT@arc@:n { o }

```

```

280 \cs_new_protected:Npn \@@_set_CT@drsc@:n #1
281 {
282 \tl_if_head_eq_meaning:nNTF { #1 } [
283 { \cs_set:Npn \CT@drsc@ { \color #1 } }
284 { \cs_set:Npn \CT@drsc@ { \color { #1 } } }
285 ]
286 \cs_generate_variant:Nn \@@_set_CT@drsc@:n { o }

```

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

```

287 \cs_new:Npn \@@_exp_color_arg:Nn #1 #2
288 {
289 \tl_if_head_eq_meaning:nNTF { #2 } [
290 { #1 #2 }
291 { #1 { #2 } }
292 ]
293 \cs_generate_variant:Nn \@@_exp_color_arg:Nn { N o }

```

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

```

294 \cs_new_protected:Npn \@@_color:n #1
295   { \tl_if_blank:nF { #1 } { \@@_exp_color_arg:Nn \color { #1 } } }
296 \cs_generate_variant:Nn \@@_color:n { o }

297 \cs_set_eq:NN \@@_old_pgfpaintanchor \pgfpaintanchor

298 \cs_new_protected:Npn \@@_rescan_for_spanish:N #1
299   {
300     \tl_set_rescan:Nno
301       #1
302     {
303       \char_set_catcode_other:N >
304       \char_set_catcode_other:N <
305     }
306     #1
307   }

```

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.

```

308 \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).

```

309 \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.

```

310 \NewExpandableDocumentCommand \NiceMatrixLastEnv { }
311   { \int_use:N \g_@@_env_int }

```

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

```

312 \cs_new_protected:Npn \@@_qpoint:n #1
313   { \pgfpaintanchor { \@@_env: - #1 } { center } }

```

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

```

314 \bool_new:N \l_@@_tabular_bool

```

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

```

315 \bool_new:N \g_@@_delims_bool
316 \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.

```
317 \bool_new:N \l_@@_preamble_bool
318 \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.

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

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

```
320 \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.

```
321 \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).

```
322 \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.).

```
323 \dim_new:N \l_@@_col_width_dim
324 \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.

```
325 \int_new:N \g_@@_row_total_int
326 \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).

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

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

```
328 \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.

```
329 \tl_new:N \l_@@_hpos_cell_tl
330 \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`.

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

Idem for the mono-row blocks.

```
332 \dim_new:N \g_@@_blocks_ht_dim
333 \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}`).

```
334 \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`.

```
335 \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.

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

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

```
337 \bool_new:N \l_@@_notes_detect_duplicates_bool
338 \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.

```
339 \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).

```
340 \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).

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

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

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

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

```
343 \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.

```
344 \bool_new:N \l_@@_X_bool
345 \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 }`).

```
346 \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.

```
347 \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.

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

349 \tl_new:N \g_@@_left_delim_tl
350 \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}`).

```
351 \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`).

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

For `\multicolumn`.

```
353 \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`.

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

```
356 \tl_new:N \l_@@_xdots_down_tl
357 \tl_new:N \l_@@_xdots_up_tl
358 \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`).

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

```
360 \cs_new_protected:Npn \@@_test_if_math_mode:
361 {
362   \if_mode_math: \else:
363     \@@_fatal:n { Outside-math-mode }
364   \fi:
365 }
```

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.

```
366 \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”.

```
367 \colorlet { nicematrix-last-col } { . }
368 \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`).

```
369 \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*.

```
370 \tl_new:N \g_@@_com_or_env_str
371 \tl_gset:Nn \g_@@_com_or_env_str { environment }
```

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

```
373 \cs_new:Npn \@@_full_name_env:
374 {
375   \str_if_eq:VnTF \g_@@_com_or_env_str { command }
376     { command \space \c_backslash_str \g_@@_name_env_str }
377     { environment \space \{ \g_@@_name_env_str \} }
378 }
```

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

```
379 \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.

```
380 \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.

```
381 \tl_new:N \g_@@_pre_code_before_tl
382 \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.

```
383 \tl_new:N \g_@@_pre_code_after_tl
384 \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.

```
385 \bool_new:N \l_@@_in_code_after_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.

```
386 \int_new:N \l_@@_old_iRow_int
387 \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).

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

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

```
389 \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.

```
390 \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.

```
391 \bool_new:N \l_@@_X_columns_aux_bool
392 \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.

```
393 \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).

```
394 \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).

```
395 \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`).

```
396 \tl_new:N \l_@@_code_before_tl
397 \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).

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

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

```
399 \dim_new:N \l_@@_x_initial_dim
400 \dim_new:N \l_@@_y_initial_dim
401 \dim_new:N \l_@@_x_final_dim
402 \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.

```
403 \dim_zero_new:N \l_@@_tmpc_dim
404 \dim_zero_new:N \l_@@_tmpd_dim
```

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

```
405 \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”.

```
406 \dim_new:N \g_@@_width_last_col_dim
407 \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.

```
408 \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!).

```
409 \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}`.

```
410 \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.

```
411 \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.

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

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

```
413 \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.

```
414 \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).

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

```
416 \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`).

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

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

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

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

The following counters will be used when drawing the rules.

```

421 \int_new:N \l_@@_start_int
422 \int_set_eq:NN \l_@@_start_int \c_one_int
423 \int_new:N \l_@@_end_int
424 \int_new:N \l_@@_local_start_int
425 \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.

```

426 \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).

```

427 \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`.

```

428 \tl_new:N \l_@@_fill_tl
429 \tl_new:N \l_@@_opacity_tl
430 \tl_new:N \l_@@_draw_tl
431 \seq_new:N \l_@@_tikz_seq
432 \clist_new:N \l_@@_borders_clist
433 \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.

```

434 \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`.

```

435 \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.

```

436 \dim_new:N \l_@@_offset_dim

```

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

```

437 \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`).

```

438 \str_new:N \l_@@_hpos_block_str
439 \str_set:Nn \l_@@_hpos_block_str { c }
440 \bool_new:N \l_@@_hpos_of_block_cap_bool

```

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

```

441 \bool_new:N \@@_nocolor_used_bool

```

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

```

442 \str_new:N \l_@@_vpos_block_str
443 \str_set:Nn \l_@@_vpos_block_str { c }

```

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

```
444 \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).

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

```
446 \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.

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

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

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

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

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

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

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

```
454 \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}`).

```
455 \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.

```
456 \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`).

```
457 \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.

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

```
459 \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.

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

```
461 \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).

```
462 \int_new:N \l_@@_last_row_int
```

```
463 \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

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

Idem for `\l_@@_last_col_without_value_bool`

```
465 \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`.

```
466 \int_new:N \l_@@_last_col_int
467 \int_set:Nn \l_@@_last_col_int { -2 }
```

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

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

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

```
468 \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`).

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

Some utilities

```
470 \cs_set_protected:Npn \@@_cut_on_hyphen:w #1-#2\q_stop
471 {
472   \cs_set_nopar:Npn \l_tmpa_tl { #1 }
473   \cs_set_nopar:Npn \l_tmpb_tl { #2 }
474 }
```

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.

```
475 \cs_new_protected:Npn \@@_expand_clist:N #1
476 {
477   \clist_if_in:NVF #1 \c_@@_all_tl
478   {
479     \clist_clear:N \l_tmpa_clist
480     \clist_map_inline:Nn #1
```

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.

```

481     {
482       \tl_if_in:nnTF { ##1 } { - }
483       { \@@_cut_on_hyphen:w ##1 \q_stop }
484       {
485         \cs_set_nopar:Npn \l_tmpa_tl { ##1 }
486         \cs_set_nopar:Npn \l_tmpb_tl { ##1 }
487       }
488       \int_step_inline:nnn { \l_tmpa_tl } { \l_tmpb_tl }
489       { \clist_put_right:Nn \l_tmpa_clist { ####1 } }
490     }
491     \tl_set_eq:NN #1 \l_tmpa_clist
492   }
493 }

```

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.

```

494 \hook_gput_code:nnn { begindocument } { . }
495 {
496   \dim_const:Nn \c_@@_shift_Ldots_last_row_dim { 0.5 em }
497   \dim_const:Nn \c_@@_shift_exterior_Vdots_dim { 0.6 em }
498   \dim_const:Nn \c_@@_innersep_middle_dim { 0.17 em }
499 }

```

6 The command `\tabularnote`

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

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

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

- During the composition of the caption (value of `\l_@@_caption_tl`), the tabular notes will be numbered from 1 to `\g_@@_notes_caption_int` and the notes themselves will be stored in `\g_@@_notes_in_caption_seq`. The structure of the components of that sequence will be the same as for `\g_@@_notes_seq`.
- After the composition of the main tabular and after the composition of the caption, the sequences `\g_@@_notes_in_caption_seq` and `\g_@@_notes_seq` will be merged (in that order) and the notes will be composed.

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

```

500 \newcounter { tabularnote }
501 \seq_new:N \g_@@_notes_seq
502 \seq_new:N \g_@@_notes_in_caption_seq

```

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

```

503 \tl_new:N \g_@@_tabularnote_tl

```

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

```

504 \seq_new:N \l_@@_notes_labels_seq
505 \newcounter{nicematrix_draft}
506 \cs_new_protected:Npn \@@_notes_format:n #1
507 {
508   \setcounter { nicematrix_draft } { #1 }
509   \@@_notes_style:n { nicematrix_draft }
510 }

```

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

```

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

```

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

```

512 \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`.

```

513 \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`.

```

514 \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`).

```

515 \hook_gput_code:nnn { begindocument } { . }
516 {
517   \IfPackageLoadedTF { enumitem }
518   {

```

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.

```

519     \newlist { tabularnotes } { enumerate } { 1 }
520     \setlist [ tabularnotes ]
521     {
522         topsep = Opt ,
523         noitemsep ,
524         leftmargin = * ,
525         align = left ,
526         labelsep = Opt ,
527         label =
528             \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotesi } } ,
529     }
530     \newlist { tabularnotes* } { enumerate* } { 1 }
531     \setlist [ tabularnotes* ]
532     {
533         afterlabel = \nobreak ,
534         itemjoin = \quad ,
535         label =
536             \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotes*i } }
537     }

```

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`.

```

538     \NewDocumentCommand \tabularnote { o m }
539     {
540         \bool_lazy_or:nnT { \cs_if_exist_p:N \@capttype } \l_@@_in_env_bool
541         {
542             \bool_lazy_and:nnTF { ! \l_@@_tabular_bool } \l_@@_in_env_bool
543             { \@@_error:n { tabularnote~forbidden } }
544             {
545                 \bool_if:NTF \l_@@_in_caption_bool
546                 \@@_tabularnote_caption:nn
547                 \@@_tabularnote:nn
548                 { #1 } { #2 }
549             }
550         }
551     }
552 }
553 {
554     \NewDocumentCommand \tabularnote { o m }
555     {
556         \@@_error_or_warning:n { enumitem~not~loaded }
557         \@@_gredirect_none:n { enumitem~not~loaded }
558     }
559 }
560 }

561 \cs_new_protected:Npn \@@_test_first_novalue:nnn #1 #2 #3
562 { \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`.

```

563 \cs_new_protected:Npn \@@_tabularnote:nn #1 #2
564 {

```

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).

```

565     \int_zero:N \l_tmpa_int
566     \bool_if:NT \l_@@_notes_detect_duplicates_bool
567     {

```

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

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

If the user have used `\tabulernote` 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@tabulernote`.

```

568     \int_zero:N \l_tmpb_int
569     \seq_map_indexed_inline:Nn \g_@@_notes_seq
570     {
571         \@@_test_first_novalue:nnn ##2 { \int_incr:N \l_tmpb_int }
572         \tl_if_eq:nnT { { #1 } { #2 } } { { ##2 }
573         {
574             \tl_if_novalue:nTF { #1 }
575             { \int_set_eq:NN \l_tmpa_int \l_tmpb_int }
576             { \int_set:Nn \l_tmpa_int { ##1 } }
577             \seq_map_break:
578         }
579     }
580     \int_if_zero:nF \l_tmpa_int
581     { \int_add:Nn \l_tmpa_int \g_@@_notes_caption_int }
582 }
583 \int_if_zero:nT \l_tmpa_int
584 {
585     \seq_gput_right:Nn \g_@@_notes_seq { { #1 } { #2 } }
586     \tl_if_novalue:nT { #1 } { \int_gincr:N \c@tabulernote }
587 }
588 \seq_put_right:Nx \l_@@_notes_labels_seq
589 {
590     \tl_if_novalue:nTF { #1 }
591     {
592         \@@_notes_format:n
593         {
594             \int_eval:n
595             {
596                 \int_if_zero:nTF \l_tmpa_int
597                 \c@tabulernote
598                 \l_tmpa_int
599             }
600         }
601     }
602     { #1 }
603 }
604 \peek_meaning:NF \tabulernote
605 {

```

If the following token is *not* a `\tabulernote`, we have finished the sequence of successive commands `\tabulernote` 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`.

```

606     \hbox_set:Nn \l_tmpa_box
607     {

```

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

```

608         \@@_notes_label_in_tabular:n
609         {

```

```

610         \seq_use:Nnnn
611         \l_@@_notes_labels_seq { , } { , } { , }
612     }
613 }

```

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

```

614     \int_gdecr:N \c@tabularnote
615     \int_set_eq:NN \l_tmpa_int \c@tabularnote
616     \refstepcounter { tabularnote }
617     \int_compare:nNnT \l_tmpa_int = \c@tabularnote
618     { \int_gincr:N \c@tabularnote }
619     \seq_clear:N \l_@@_notes_labels_seq
620     \bool_lazy_or:nnTF
621     { \tl_if_eq_p:NN \l_@@_hpos_cell_tl \c_@@_c_tl }
622     { \tl_if_eq_p:NN \l_@@_hpos_cell_tl \c_@@_r_tl }
623     {
624         \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).

```

625         \skip_horizontal:n { \box_wd:N \l_tmpa_box }
626     }
627     { \box_use:N \l_tmpa_box }
628 }
629 }

```

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`.

```

630 \cs_new_protected:Npn \g_@@_tabularnote_caption:nn #1 #2
631 {
632     \bool_if:NTF \g_@@_caption_finished_bool
633     {
634         \int_compare:nNnT \c@tabularnote = \g_@@_notes_caption_int
635         { \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`!

```

636         \seq_if_in:NnF \g_@@_notes_in_caption_seq { { #1 } { #2 } }
637         { \@@_error:n { Identical-notes-in-caption } }
638     }
639     {

```

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

```

640         \seq_if_in:NnTF \g_@@_notes_in_caption_seq { { #1 } { #2 } }
641         {

```

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.

```

642         \bool_gset_true:N \g_@@_caption_finished_bool
643         \int_gset_eq:NN \g_@@_notes_caption_int \c@tabularnote
644         \int_gzero:N \c@tabularnote
645     }
646     { \seq_gput_right:Nn \g_@@_notes_in_caption_seq { { #1 } { #2 } } }
647 }

```

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!

```

648 \tl_if_novalue:nT { #1 } { \int_gincr:N \c@tabularnote }
649 \seq_put_right:Nx \l_@@_notes_labels_seq
650 {
651   \tl_if_novalue:nTF { #1 }
652     { \@@_notes_format:n { \int_use:N \c@tabularnote } }
653     { #1 }
654 }
655 \peek_meaning:NF \tabularnote
656 {
657   \@@_notes_label_in_tabular:n
658     { \seq_use:Nnnn \l_@@_notes_labels_seq { , } { , } { , } }
659   \seq_clear:N \l_@@_notes_labels_seq
660 }
661 }

662 \cs_new_protected:Npn \@@_count_novalue_first:nn #1 #2
663 { \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.

```

664 \cs_new_protected:Npn \@@_pgf_rect_node:nnnnn #1 #2 #3 #4 #5
665 {
666   \begin { pgfscope }
667   \pgfset
668   {
669     inner~sep = \c_zero_dim ,
670     minimum~size = \c_zero_dim
671   }
672   \pgftransformshift { \pgfpoint { 0.5 * ( #2 + #4 ) } { 0.5 * ( #3 + #5 ) } }
673   \pgfnode
674   { rectangle }
675   { center }
676   {
677     \vbox_to_ht:nn
678     { \dim_abs:n { #5 - #3 } }
679     {
680       \vfill
681       \hbox_to_wd:nn { \dim_abs:n { #4 - #2 } } { }
682     }
683   }
684   { #1 }
685   { }
686   \end { pgfscope }
687 }

```

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.

```

688 \cs_new_protected:Npn \@@_pgf_rect_node:nnn #1 #2 #3
689 {
690   \begin { pgfscope }
691   \pgfset
692   {
693     inner~sep = \c_zero_dim ,
694     minimum~size = \c_zero_dim

```

```

695     }
696     \pgftransformshift { \pgfpointscale { 0.5 } { \pgfpointadd { #2 } { #3 } } }
697     \pgfpointdiff { #3 } { #2 }
698     \pgfgetlastxy \l_tmpa_dim \l_tmpb_dim
699     \pgfnode
700     { rectangle }
701     { center }
702     {
703         \vbox_to_ht:nn
704         { \dim_abs:n \l_tmpb_dim }
705         { \vfill \hbox_to_wd:nn { \dim_abs:n \l_tmpa_dim } { } }
706     }
707     { #1 }
708     { }
709     \end { pgfscope }
710 }

```

8 The options

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

```

711 \tl_new:N \l_@@_caption_tl
712 \tl_new:N \l_@@_short_caption_tl
713 \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).

```

714 \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.

```

715 \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`.

```

716 \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`).

```

717 \dim_new:N \l_@@_cell_space_top_limit_dim
718 \dim_new:N \l_@@_cell_space_bottom_limit_dim

```

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

```

719 \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.

```

720 \dim_new:N \l_@@_xdots_inter_dim
721 \hook_gput_code:nnn { begindocument } { . }
722 { \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).

```

723 \dim_new:N \l_@@_xdots_shorten_start_dim
724 \dim_new:N \l_@@_xdots_shorten_end_dim
725 \hook_gput_code:nnn { begindocument } { . }
726 {
727   \dim_set:Nn \l_@@_xdots_shorten_start_dim { 0.3 em }
728   \dim_set:Nn \l_@@_xdots_shorten_end_dim { 0.3 em }
729 }
```

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.

```

730 \dim_new:N \l_@@_xdots_radius_dim
731 \hook_gput_code:nnn { begindocument } { . }
732 { \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.

```

733 \tl_new:N \l_@@_xdots_line_style_tl
734 \tl_const:Nn \c_@@_standard_tl { standard }
735 \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`.

```

736 \bool_new:N \l_@@_light_syntax_bool
737 \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).

```

738 \tl_new:N \l_@@_baseline_tl
739 \tl_set:Nn \l_@@_baseline_tl { c }
```

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`).

```

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

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

```

741 \bool_new:N \l_@@_parallelize_diags_bool
742 \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.

```

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

```

744 \dim_new:N \l_@@_notes_above_space_dim
745 \hook_gput_code:nnn { begindocument } { . }
746 { \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.).

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

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

```
748 \cs_new_protected:Npn \@@_reset_arraystretch:
749 { \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).

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

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

```
751 \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.

```
752 \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”.

```
753 \bool_new:N \l_@@_medium_nodes_bool
754 \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).

```
755 \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`).

```
756 \dim_new:N \l_@@_left_margin_dim
757 \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`.

```
758 \dim_new:N \l_@@_extra_left_margin_dim
759 \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.

```
760 \tl_new:N \l_@@_end_of_row_tl
761 \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 “:”.

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

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

```
763 \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 function 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.

```

764 \bool_new:N \l_@@_delimiters_max_width_bool

765 \keys_define:nn { NiceMatrix / xdots }
766 {
767   shorten-start .code:n =
768     \hook_gput_code:nnn { begindocument } { . }
769     { \dim_set:Nn \l_@@_xdots_shorten_start_dim { #1 } } ,
770   shorten-end .code:n =
771     \hook_gput_code:nnn { begindocument } { . }
772     { \dim_set:Nn \l_@@_xdots_shorten_end_dim { #1 } } ,
773   shorten-start .value_required:n = true ,
774   shorten-end .value_required:n = true ,
775   shorten .code:n =
776     \hook_gput_code:nnn { begindocument } { . }
777     {
778       \dim_set:Nn \l_@@_xdots_shorten_start_dim { #1 }
779       \dim_set:Nn \l_@@_xdots_shorten_end_dim { #1 }
780     } ,
781   shorten .value_required:n = true ,
782   horizontal-labels .bool_set:N = \l_@@_xdots_h_labels_bool ,
783   horizontal-labels .default:n = true ,
784   line-style .code:n =
785     {
786       \bool_lazy_or:nnTF
787         { \cs_if_exist_p:N \tikzpicture }
788         { \str_if_eq_p:nn { #1 } { standard } }
789         { \tl_set:Nn \l_@@_xdots_line_style_tl { #1 } }
790         { @@_error:n { bad-option-for~line-style } }
791     } ,
792   line-style .value_required:n = true ,
793   color .tl_set:N = \l_@@_xdots_color_tl ,
794   color .value_required:n = true ,
795   radius .code:n =
796     \hook_gput_code:nnn { begindocument } { . }
797     { \dim_set:Nn \l_@@_xdots_radius_dim { #1 } } ,
798   radius .value_required:n = true ,
799   inter .code:n =
800     \hook_gput_code:nnn { begindocument } { . }
801     { \dim_set:Nn \l_@@_xdots_inter_dim { #1 } } ,
802   radius .value_required:n = true ,

```

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

```

803   down .code:n = \tl_put_right:Nn \l_@@_xdots_down_tl { #1 } ,
804   up .code:n = \tl_put_right:Nn \l_@@_xdots_up_tl { #1 } ,
805   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).

```

806   draw-first .code:n = \prg_do_nothing: ,
807   unknown .code:n = @@_error:n { Unknown~key~for~xdots }
808 }

```

```

809 \keys_define:nn { NiceMatrix / rules }
810 {
811   color .tl_set:N = \l_@@_rules_color_tl ,
812   color .value_required:n = true ,
813   width .dim_set:N = \arrayrulewidth ,
814   width .value_required:n = true ,
815   unknown .code:n = \@@_error:n { Unknown~key~for~rules }
816 }

```

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

```

817 \keys_define:nn { NiceMatrix / Global }
818 {
819   no-cell-nodes .code:n =
820     \cs_set_protected:Npn \@@_node_for_cell:
821     { \box_use_drop:N \l_@@_cell_box } ,
822   no-cell-nodes .value_forbidden:n = true ,
823   rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
824   rounded-corners .default:n = 4 pt ,
825   custom-line .code:n = \@@_custom_line:n { #1 } ,
826   rules .code:n = \keys_set:nn { NiceMatrix / rules } { #1 } ,
827   rules .value_required:n = true ,
828   standard-cline .bool_set:N = \l_@@_standard_cline_bool ,
829   standard-cline .default:n = true ,
830   cell-space-top-limit .dim_set:N = \l_@@_cell_space_top_limit_dim ,
831   cell-space-top-limit .value_required:n = true ,
832   cell-space-bottom-limit .dim_set:N = \l_@@_cell_space_bottom_limit_dim ,
833   cell-space-bottom-limit .value_required:n = true ,
834   cell-space-limits .meta:n =
835   {
836     cell-space-top-limit = #1 ,
837     cell-space-bottom-limit = #1 ,
838   } ,
839   cell-space-limits .value_required:n = true ,
840   xdots .code:n = \keys_set:nn { NiceMatrix / xdots } { #1 } ,
841   light-syntax .code:n =
842     \bool_set_true:N \l_@@_light_syntax_bool
843     \bool_set_false:N \l_@@_light_syntax_expanded_bool ,
844   light-syntax .value_forbidden:n = true ,
845   light-syntax-expanded .code:n =
846     \bool_set_true:N \l_@@_light_syntax_bool
847     \bool_set_true:N \l_@@_light_syntax_expanded_bool ,
848   light-syntax-expanded .value_forbidden:n = true ,
849   end-of-row .tl_set:N = \l_@@_end_of_row_tl ,
850   end-of-row .value_required:n = true ,
851   first-col .code:n = \int_zero:N \l_@@_first_col_int ,
852   first-row .code:n = \int_zero:N \l_@@_first_row_int ,
853   last-row .int_set:N = \l_@@_last_row_int ,
854   last-row .default:n = -1 ,
855   code-for-first-col .tl_set:N = \l_@@_code_for_first_col_tl ,
856   code-for-first-col .value_required:n = true ,
857   code-for-last-col .tl_set:N = \l_@@_code_for_last_col_tl ,
858   code-for-last-col .value_required:n = true ,
859   code-for-first-row .tl_set:N = \l_@@_code_for_first_row_tl ,
860   code-for-first-row .value_required:n = true ,
861   code-for-last-row .tl_set:N = \l_@@_code_for_last_row_tl ,
862   code-for-last-row .value_required:n = true ,
863   hlines .clist_set:N = \l_@@_hlines_clist ,
864   vlines .clist_set:N = \l_@@_vlines_clist ,
865   hlines .default:n = all ,
866   vlines .default:n = all ,
867   vlines-in-sub-matrix .code:n =
868   {

```

```

869 \tl_if_single_token:nTF { #1 }
870 {
871     \tl_if_in:NnTF \c_@@_forbidden_letters_tl { #1 }
872     { \@@_error:nn { Forbidden~letter } { #1 } }

```

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

```

873     { \cs_set_eq:cN { @@_#1 } \@@_make_preamble_vlism:n }
874 }
875 { \@@_error:n { One~letter~allowed } }
876 } ,
877 vlines-in-sub-matrix .value_required:n = true ,
878 hvlines .code:n =
879 {
880     \bool_set_true:N \l_@@_hvlines_bool
881     \tl_set_eq:NN \l_@@_vlines_clist \c_@@_all_tl
882     \tl_set_eq:NN \l_@@_hlines_clist \c_@@_all_tl
883 } ,
884 hvlines-except-borders .code:n =
885 {
886     \tl_set_eq:NN \l_@@_vlines_clist \c_@@_all_tl
887     \tl_set_eq:NN \l_@@_hlines_clist \c_@@_all_tl
888     \bool_set_true:N \l_@@_hvlines_bool
889     \bool_set_true:N \l_@@_except_borders_bool
890 } ,
891 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.

```

892 renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
893 renew-dots .value_forbidden:n = true ,
894 nullify-dots .bool_set:N = \l_@@_nullify_dots_bool ,
895 create-medium-nodes .bool_set:N = \l_@@_medium_nodes_bool ,
896 create-large-nodes .bool_set:N = \l_@@_large_nodes_bool ,
897 create-extra-nodes .meta:n =
898 { create-medium-nodes , create-large-nodes } ,
899 left-margin .dim_set:N = \l_@@_left_margin_dim ,
900 left-margin .default:n = \arraycolsep ,
901 right-margin .dim_set:N = \l_@@_right_margin_dim ,
902 right-margin .default:n = \arraycolsep ,
903 margin .meta:n = { left-margin = #1 , right-margin = #1 } ,
904 margin .default:n = \arraycolsep ,
905 extra-left-margin .dim_set:N = \l_@@_extra_left_margin_dim ,
906 extra-right-margin .dim_set:N = \l_@@_extra_right_margin_dim ,
907 extra-margin .meta:n =
908 { extra-left-margin = #1 , extra-right-margin = #1 } ,
909 extra-margin .value_required:n = true ,
910 respect-arraystretch .code:n =
911 \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
912 respect-arraystretch .value_forbidden:n = true ,
913 pgf-node-code .tl_set:N = \l_@@_pgf_node_code_tl ,
914 pgf-node-code .value_required:n = true
915 }

```

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

```

916 \keys_define:nn { NiceMatrix / Env }
917 {
918     corners .clist_set:N = \l_@@_corners_clist ,
919     corners .default:n = { NW , SW , NE , SE } ,
920     code-before .code:n =
921     {
922         \tl_if_empty:nF { #1 }

```

```

923     {
924         \tl_gput_left:Nn \g_@@_pre_code_before_tl { #1 }
925         \bool_set_true:N \l_@@_code_before_bool
926     }
927 } ,
928 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}`.

```

929 c .code:n = \tl_set:Nn \l_@@_baseline_tl c ,
930 t .code:n = \tl_set:Nn \l_@@_baseline_tl t ,
931 b .code:n = \tl_set:Nn \l_@@_baseline_tl b ,
932 baseline .tl_set:N = \l_@@_baseline_tl ,
933 baseline .value_required:n = true ,
934 columns-width .code:n =
935     \tl_if_eq:nnTF { #1 } { auto }
936     { \bool_set_true:N \l_@@_auto_columns_width_bool }
937     { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,
938 columns-width .value_required:n = true ,
939 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.

```

940     \legacy_if:nF { measuring@ }
941     {
942         \str_set:Nx \l_tmpa_str { #1 }
943         \seq_if_in:NVTF \g_@@_names_seq \l_tmpa_str
944         { \@@_error:nn { Duplicate~name } { #1 } }
945         { \seq_gput_left:NV \g_@@_names_seq \l_tmpa_str }
946         \str_set_eq:NN \l_@@_name_str \l_tmpa_str
947     } ,
948 name .value_required:n = true ,
949 code-after .tl_gset:N = \g_nicematrix_code_after_tl ,
950 code-after .value_required:n = true ,
951 color-inside .code:n =
952     \bool_set_true:N \l_@@_color_inside_bool
953     \bool_set_true:N \l_@@_code_before_bool ,
954 color-inside .value_forbidden:n = true ,
955 colortbl-like .meta:n = color-inside
956 }
957 \keys_define:nn { NiceMatrix / notes }
958 {
959     para .bool_set:N = \l_@@_notes_para_bool ,
960     para .default:n = true ,
961     code-before .tl_set:N = \l_@@_notes_code_before_tl ,
962     code-before .value_required:n = true ,
963     code-after .tl_set:N = \l_@@_notes_code_after_tl ,
964     code-after .value_required:n = true ,
965     bottomrule .bool_set:N = \l_@@_notes_bottomrule_bool ,
966     bottomrule .default:n = true ,
967     style .cs_set:Np = \@@_notes_style:n #1 ,
968     style .value_required:n = true ,
969     label-in-tabular .cs_set:Np = \@@_notes_label_in_tabular:n #1 ,
970     label-in-tabular .value_required:n = true ,
971     label-in-list .cs_set:Np = \@@_notes_label_in_list:n #1 ,
972     label-in-list .value_required:n = true ,
973     enumitem-keys .code:n =
974     {
975         \hook_gput_code:nnn { begindocument } { . }
976         {
977             \IfPackageLoadedTF { enumitem }
978             { \setlist* [ tabularnotes ] { #1 } }
979             { }

```

```

980     }
981   } ,
982   enumitem-keys .value_required:n = true ,
983   enumitem-keys-para .code:n =
984   {
985     \hook_gput_code:nnn { begindocument } { . }
986     {
987       \IfPackageLoadedTF { enumitem }
988       { \setlist* [ tabularnotes* ] { #1 } }
989       { }
990     }
991   } ,
992   enumitem-keys-para .value_required:n = true ,
993   detect-duplicates .bool_set:N = \l_@@_notes_detect_duplicates_bool ,
994   detect-duplicates .default:n = true ,
995   unknown .code:n = \@@_error:n { Unknown~key~for~notes }
996 }

997 \keys_define:nn { NiceMatrix / delimiters }
998 {
999   max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1000   max-width .default:n = true ,
1001   color .tl_set:N = \l_@@_delimiters_color_tl ,
1002   color .value_required:n = true ,
1003 }

```

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

```

1004 \keys_define:nn { NiceMatrix }
1005 {
1006   NiceMatrixOptions .inherit:n =
1007   { NiceMatrix / Global } ,
1008   NiceMatrixOptions / xdots .inherit:n = NiceMatrix / xdots ,
1009   NiceMatrixOptions / rules .inherit:n = NiceMatrix / rules ,
1010   NiceMatrixOptions / notes .inherit:n = NiceMatrix / notes ,
1011   NiceMatrixOptions / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
1012   SubMatrix / rules .inherit:n = NiceMatrix / rules ,
1013   CodeAfter / xdots .inherit:n = NiceMatrix / xdots ,
1014   CodeBefore / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
1015   CodeAfter / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
1016   NiceMatrix .inherit:n =
1017   {
1018     NiceMatrix / Global ,
1019     NiceMatrix / Env ,
1020   } ,
1021   NiceMatrix / xdots .inherit:n = NiceMatrix / xdots ,
1022   NiceMatrix / rules .inherit:n = NiceMatrix / rules ,
1023   NiceTabular .inherit:n =
1024   {
1025     NiceMatrix / Global ,
1026     NiceMatrix / Env
1027   } ,
1028   NiceTabular / xdots .inherit:n = NiceMatrix / xdots ,
1029   NiceTabular / rules .inherit:n = NiceMatrix / rules ,
1030   NiceTabular / notes .inherit:n = NiceMatrix / notes ,
1031   NiceArray .inherit:n =
1032   {
1033     NiceMatrix / Global ,
1034     NiceMatrix / Env ,
1035   } ,
1036   NiceArray / xdots .inherit:n = NiceMatrix / xdots ,
1037   NiceArray / rules .inherit:n = NiceMatrix / rules ,
1038   pNiceArray .inherit:n =

```

```

1039     {
1040         NiceMatrix / Global ,
1041         NiceMatrix / Env ,
1042     } ,
1043     pNiceArray / xdots .inherit:n = NiceMatrix / xdots ,
1044     pNiceArray / rules .inherit:n = NiceMatrix / rules ,
1045 }

```

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

```

1046 \keys_define:nn { NiceMatrix / NiceMatrixOptions }
1047 {
1048     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1049     delimiters / color .value_required:n = true ,
1050     delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1051     delimiters / max-width .default:n = true ,
1052     delimiters .code:n = \keys_set:nn { NiceMatrix / delimiters } { #1 } ,
1053     delimiters .value_required:n = true ,
1054     width .dim_set:N = \l_@@_width_dim ,
1055     width .value_required:n = true ,
1056     last-col .code:n =
1057         \tl_if_empty:nF { #1 }
1058         { \@@_error:n { last-col~non-empty~for~NiceMatrixOptions } }
1059         \int_zero:N \l_@@_last_col_int ,
1060     small .bool_set:N = \l_@@_small_bool ,
1061     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.

```

1062     renew-matrix .code:n = \@@_renew_matrix: ,
1063     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}`.

```

1064     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.

```

1065     columns-width .code:n =
1066         \tl_if_eq:nnTF { #1 } { auto }
1067         { \@@_error:n { Option~auto~for~columns-width } }
1068         { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,

```

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

```

1069     allow-duplicate-names .code:n =
1070         \@@_msg_redirect_name:nn { Duplicate~name } { none } ,
1071     allow-duplicate-names .value_forbidden:n = true ,
1072     notes .code:n = \keys_set:nn { NiceMatrix / notes } { #1 } ,
1073     notes .value_required:n = true ,
1074     sub-matrix .code:n = \keys_set:nn { NiceMatrix / sub-matrix } { #1 } ,
1075     sub-matrix .value_required:n = true ,
1076     matrix / columns-type .tl_set:N = \l_@@_columns_type_tl ,
1077     matrix / columns-type .value_required:n = true ,
1078     caption-above .bool_set:N = \l_@@_caption_above_bool ,
1079     caption-above .default:n = true ,
1080     unknown .code:n = \@@_error:n { Unknown~key~for~NiceMatrixOptions }
1081 }

```

`\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.

```
1082 \NewDocumentCommand \NiceMatrixOptions { m }
1083 { \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.

```
1084 \keys_define:nn { NiceMatrix / NiceMatrix }
1085 {
1086   last-col .code:n = \tl_if_empty:nTF { #1 }
1087   {
1088     \bool_set_true:N \l_@@_last_col_without_value_bool
1089     \int_set:Nn \l_@@_last_col_int { -1 }
1090   }
1091   { \int_set:Nn \l_@@_last_col_int { #1 } } ,
1092   columns-type .tl_set:N = \l_@@_columns_type_tl ,
1093   columns-type .value_required:n = true ,
1094   l .meta:n = { columns-type = l } ,
1095   r .meta:n = { columns-type = r } ,
1096   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1097   delimiters / color .value_required:n = true ,
1098   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1099   delimiters / max-width .default:n = true ,
1100   delimiters .code:n = \keys_set:nn { NiceMatrix / delimiters } { #1 } ,
1101   delimiters .value_required:n = true ,
1102   small .bool_set:N = \l_@@_small_bool ,
1103   small .value_forbidden:n = true ,
1104   unknown .code:n = \@@_error:n { Unknown~key~for~NiceMatrix }
1105 }
```

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

```
1106 \keys_define:nn { NiceMatrix / NiceArray }
1107 {
```

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.

```
1108   small .bool_set:N = \l_@@_small_bool ,
1109   small .value_forbidden:n = true ,
1110   last-col .code:n = \tl_if_empty:nF { #1 }
1111   { \@@_error:n { last-col-non-empty-for-NiceArray } }
1112   \int_zero:N \l_@@_last_col_int ,
1113   r .code:n = \@@_error:n { r-or-l-with-preamble } ,
1114   l .code:n = \@@_error:n { r-or-l-with-preamble } ,
1115   unknown .code:n = \@@_error:n { Unknown~key~for~NiceArray }
1116 }

1117 \keys_define:nn { NiceMatrix / pNiceArray }
1118 {
1119   first-col .code:n = \int_zero:N \l_@@_first_col_int ,
1120   last-col .code:n = \tl_if_empty:nF { #1 }
1121   { \@@_error:n { last-col-non-empty-for-NiceArray } }
1122   \int_zero:N \l_@@_last_col_int ,
1123   first-row .code:n = \int_zero:N \l_@@_first_row_int ,
1124   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1125   delimiters / color .value_required:n = true ,
1126   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1127   delimiters / max-width .default:n = true ,
1128   delimiters .code:n = \keys_set:nn { NiceMatrix / delimiters } { #1 } ,
1129   delimiters .value_required:n = true ,
1130   small .bool_set:N = \l_@@_small_bool ,
1131   small .value_forbidden:n = true ,
```

```

1132   r .code:n = \@@_error:n { r~or~l~with~preamble } ,
1133   l .code:n = \@@_error:n { r~or~l~with~preamble } ,
1134   unknown .code:n = \@@_error:n { Unknown~key~for~NiceMatrix }
1135 }

```

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

```

1136 \keys_define:nn { NiceMatrix / NiceTabular }
1137 {

```

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.

```

1138   width .code:n = \dim_set:Nn \l_@@_width_dim { #1 }
1139   \bool_set_true:N \l_@@_width_used_bool ,
1140   width .value_required:n = true ,
1141   notes .code:n = \keys_set:nn { NiceMatrix / notes } { #1 } ,
1142   tabularnote .tl_gset:N = \g_@@_tabularnote_tl ,
1143   tabularnote .value_required:n = true ,
1144   caption .tl_set:N = \l_@@_caption_tl ,
1145   caption .value_required:n = true ,
1146   short-caption .tl_set:N = \l_@@_short_caption_tl ,
1147   short-caption .value_required:n = true ,
1148   label .tl_set:N = \l_@@_label_tl ,
1149   label .value_required:n = true ,
1150   last-col .code:n = \tl_if_empty:nF {#1}
1151   { \@@_error:n { last-col-non-empty-for-NiceArray } }
1152   \int_zero:N \l_@@_last_col_int ,
1153   r .code:n = \@@_error:n { r~or~l~with~preamble } ,
1154   l .code:n = \@@_error:n { r~or~l~with~preamble } ,
1155   unknown .code:n = \@@_error:n { Unknown~key~for~NiceTabular }
1156 }

```

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
```

```

1157 \keys_define:nn { NiceMatrix / CodeAfter }
1158 {
1159   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1160   delimiters / color .value_required:n = true ,
1161   rules .code:n = \keys_set:nn { NiceMatrix / rules } { #1 } ,
1162   rules .value_required:n = true ,
1163   xdots .code:n = \keys_set:nn { NiceMatrix / xdots } { #1 } ,
1164   sub-matrix .code:n = \keys_set:nn { NiceMatrix / sub-matrix } { #1 } ,
1165   sub-matrix .value_required:n = true ,
1166   unknown .code:n = \@@_error:n { Unknown~key~for~CodeAfter }
1167 }

```

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}).

```

1168 \cs_new_protected:Npn \@@_cell_begin:w
1169 {

```

`\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.

```
1170 \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).

```
1171 \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
```

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

```
1172 \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.

```
1173 \int_compare:nNnT \c@jCol = \c_one_int
1174 { \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:`.

```
1175 \hbox_set:Nw \l_@@_cell_box
```

The following command is nullified in the tabulars.

```
1176 \@@_tuning_not_tabular_begin:
1177 \@@_tuning_first_row:
1178 \@@_tuning_last_row:
1179 \g_@@_row_style_tl
1180 }
```

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

```
1181 \cs_new_protected:Npn \@@_tuning_first_row:
1182 {
1183   \int_if_zero:nT \c@iRow
1184   {
1185     \int_compare:nNnT \c@jCol > \c_zero_int
1186     {
1187       \l_@@_code_for_first_row_tl
1188       \xglobal \colorlet { nicematrix-first-row } { . }
1189     }
1190   }
1191 }
```

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

```
1192 \cs_new_protected:Npn \@@_tuning_last_row:
1193 {
1194   \int_compare:nNnT \c@iRow = \l_@@_last_row_int
1195   {
1196     \l_@@_code_for_last_row_tl
1197     \xglobal \colorlet { nicematrix-last-row } { . }
1198   }
1199 }
```

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

```
1200 \cs_set_eq:NN \@@_tuning_key_small: \prg_do_nothing:
```

The following commands are nullified in the tabulars.

```
1201 \cs_set_nopar:Npn \@@_tuning_not_tabular_begin:
1202 {
1203   \c_math_toggle_token
```

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

```
1204 \@@_tuning_key_small:
1205 }
1206 \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.

```

1207 \cs_new_protected:Npn \@@_begin_of_row:
1208 {
1209   \int_gincr:N \c@iRow
1210   \dim_gset_eq:NN \g_@@_dp_ante_last_row_dim \g_@@_dp_last_row_dim
1211   \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
1212   \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1213   \pgfpicture
1214   \pgfrememberpicturepositiononpagetrue
1215   \pgfcoordinate
1216   { \@@_env: - row - \int_use:N \c@iRow - base }
1217   { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
1218   \str_if_empty:NF \l_@@_name_str
1219   {
1220     \pgfnodealias
1221     { \l_@@_name_str - row - \int_use:N \c@iRow - base }
1222     { \@@_env: - row - \int_use:N \c@iRow - base }
1223   }
1224   \endpgfpicture
1225 }

```

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.

```

1226 \cs_new_protected:Npn \@@_update_for_first_and_last_row:
1227 {
1228   \int_if_zero:nTF \c@iRow
1229   {
1230     \dim_gset:Nn \g_@@_dp_row_zero_dim
1231     { \dim_max:nn \g_@@_dp_row_zero_dim { \box_dp:N \l_@@_cell_box } }
1232     \dim_gset:Nn \g_@@_ht_row_zero_dim
1233     { \dim_max:nn \g_@@_ht_row_zero_dim { \box_ht:N \l_@@_cell_box } }
1234   }
1235   {
1236     \int_compare:nNnT \c@iRow = \c_one_int
1237     {
1238       \dim_gset:Nn \g_@@_ht_row_one_dim
1239       { \dim_max:nn \g_@@_ht_row_one_dim { \box_ht:N \l_@@_cell_box } }
1240     }
1241   }
1242 }

1243 \cs_new_protected:Npn \@@_rotate_cell_box:
1244 {
1245   \box_rotate:Nn \l_@@_cell_box { 90 }
1246   \bool_if:NTF \g_@@_rotate_c_bool
1247   {
1248     \hbox_set:Nn \l_@@_cell_box
1249     {
1250       \c_math_toggle_token
1251       \vcenter { \box_use:N \l_@@_cell_box }
1252       \c_math_toggle_token
1253     }
1254   }
1255   {
1256     \int_compare:nNnT \c@iRow = \l_@@_last_row_int
1257     {
1258       \vbox_set_top:Nn \l_@@_cell_box
1259       {

```

```

1260         \vbox_to_zero:n { }
1261         \skip_vertical:n { - \box_ht:N \@arstrutbox + 0.8 ex }
1262         \box_use:N \l_@@_cell_box
1263     }
1264 }
1265 }
1266 \bool_gset_false:N \g_@@_rotate_bool
1267 \bool_gset_false:N \g_@@_rotate_c_bool
1268 }
1269 \cs_new_protected:Npn \@@_adjust_size_box:
1270 {
1271     \dim_compare:nNt \g_@@_blocks_wd_dim > \c_zero_dim
1272     {
1273         \box_set_wd:Nn \l_@@_cell_box
1274         { \dim_max:nn { \box_wd:N \l_@@_cell_box } \g_@@_blocks_wd_dim }
1275         \dim_gzero:N \g_@@_blocks_wd_dim
1276     }
1277     \dim_compare:nNt \g_@@_blocks_dp_dim > \c_zero_dim
1278     {
1279         \box_set_dp:Nn \l_@@_cell_box
1280         { \dim_max:nn { \box_dp:N \l_@@_cell_box } \g_@@_blocks_dp_dim }
1281         \dim_gzero:N \g_@@_blocks_dp_dim
1282     }
1283     \dim_compare:nNt \g_@@_blocks_ht_dim > \c_zero_dim
1284     {
1285         \box_set_ht:Nn \l_@@_cell_box
1286         { \dim_max:nn { \box_ht:N \l_@@_cell_box } \g_@@_blocks_ht_dim }
1287         \dim_gzero:N \g_@@_blocks_ht_dim
1288     }
1289 }
1290 \cs_new_protected:Npn \@@_cell_end:
1291 {

```

The following command is nullified in the tabulars.

```

1292     \@@_tuning_not_tabular_end:
1293     \hbox_set_end:
1294     \@@_cell_end_i:
1295 }
1296 \cs_new_protected:Npn \@@_cell_end_i:
1297 {

```

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.

```

1298     \g_@@_cell_after_hook_tl
1299     \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
1300     \@@_adjust_size_box:
1301     \box_set_ht:Nn \l_@@_cell_box
1302     { \box_ht:N \l_@@_cell_box + \l_@@_cell_space_top_limit_dim }
1303     \box_set_dp:Nn \l_@@_cell_box
1304     { \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”).

```

1305     \@@_update_max_cell_width:

```

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

```

1306     \@@_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.

```

1307 \bool_if:NTF \g_@@_empty_cell_bool
1308 { \box_use_drop:N \l_@@_cell_box }
1309 {
1310   \bool_if:NTF \g_@@_not_empty_cell_bool
1311   \@@_node_for_cell:
1312   {
1313     \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > \c_zero_dim
1314     \@@_node_for_cell:
1315     { \box_use_drop:N \l_@@_cell_box }
1316   }
1317 }
1318 \int_gset:Nn \g_@@_col_total_int { \int_max:nn \g_@@_col_total_int \c@jCol }
1319 \bool_gset_false:N \g_@@_empty_cell_bool
1320 \bool_gset_false:N \g_@@_not_empty_cell_bool
1321 }

```

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

```

1322 \cs_new_protected:Npn \@@_update_max_cell_width:
1323 {
1324   \dim_gset:Nn \g_@@_max_cell_width_dim
1325   { \dim_max:nn \g_@@_max_cell_width_dim { \box_wd:N \l_@@_cell_box } }
1326 }

```

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`).

```

1327 \cs_new_protected:Npn \@@_cell_end_for_w_s:
1328 {
1329   \@@_math_toggle:
1330   \hbox_set_end:
1331   \bool_if:NF \g_@@_rotate_bool
1332   {
1333     \hbox_set:Nn \l_@@_cell_box
1334     {
1335       \makebox [ \l_@@_col_width_dim ] [ s ]
1336       { \hbox_unpack_drop:N \l_@@_cell_box }
1337     }
1338   }
1339   \@@_cell_end_i:
1340 }

```

```

1341 \pgfset
1342 {
1343   nicematrix / cell-node /.style =
1344   {
1345     inner-sep = \c_zero_dim ,
1346     minimum-width = \c_zero_dim

```

```

1347 }
1348 }

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

For example, for the following matrix,

```

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

```

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

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

```

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

```

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

```

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

```

```

1413 \cs_new_protected:Npn \@@_array:
1414 {
1415   % \begin{macrocode}
1416   \dim_set:Nn \col@sep
1417     { \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
1418   \dim_compare:nNnTF \l_@@_tabular_width_dim = \c_zero_dim
1419     { \cs_set_nopar:Npn \@halignto { } }
1420     { \cs_set_nopar:Npx \@halignto { to \dim_use:N \l_@@_tabular_width_dim } }

```

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

```

1421 \@tabarray

```

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

```

1422 [ \str_if_eq:VnTF \l_@@_baseline_tl c c t ]
1423 }

```

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.

```

1424 \cs_set_eq:NN \@@_old_ialign: \ialign

```

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

```

1425 \cs_new_protected:Npn \@@_create_row_node:
1426 {
1427   \int_compare:nNnT \c@iRow > \g_@@_last_row_node_int
1428   {
1429     \int_gset_eq:NN \g_@@_last_row_node_int \c@iRow
1430     \@@_create_row_node_i:
1431   }
1432 }
1433 \cs_new_protected:Npn \@@_create_row_node_i:
1434 {

```

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

```

1435   \hbox
1436   {
1437     \bool_if:NT \l_@@_code_before_bool
1438     {
1439       \vtop
1440       {
1441         \skip_vertical:N 0.5\arrayrulewidth
1442         \pgfsys@markposition
1443         { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1444         \skip_vertical:N -0.5\arrayrulewidth
1445       }
1446     }
1447     \pgfpicture
1448     \pgfrememberpicturepositiononpagetrue
1449     \pgfcoordinate { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1450     { \pgfpoint \c_zero_dim { - 0.5 \arrayrulewidth } }
1451     \str_if_empty:NF \l_@@_name_str
1452     {
1453       \pgfnodealias
1454       { \l_@@_name_str - row - \int_eval:n { \c@iRow + 1 } }
1455       { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1456     }
1457     \endpgfpicture
1458   }
1459 }

```

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

```

1460 \cs_new:Npn \@@_everycr: { \noalign { \@@_everycr_i: } }
1461 \cs_new_protected:Npn \@@_everycr_i:
1462 {
1463   \int_gzero:N \c@jCol
1464   \bool_gset_false:N \g_@@_after_col_zero_bool
1465   \bool_if:NF \g_@@_row_of_col_done_bool
1466   {
1467     \@@_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).

```

1468   \tl_if_empty:NF \l_@@_hlines_clist
1469   {
1470     \tl_if_eq:NNF \l_@@_hlines_clist \c_@@_all_tl
1471     {
1472       \exp_args:NNe
1473       \clist_if_in:NnT
1474       \l_@@_hlines_clist
1475       { \int_eval:n { \c@iRow + 1 } }
1476     }
1477   }

```

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.

```

1478         \int_compare:nNnT \c@iRow > { -1 }
1479         {
1480             \int_compare:nNnF \c@iRow = \l_@@_last_row_int

```

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. We use a TeX group in order to limit the scope of `\CT@arc@`.

```

1481             { \hrule height \arrayrulewidth width \c_zero_dim }
1482         }
1483     }
1484 }
1485 }
1486 }

```

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

```

1487 \cs_set_protected:Npn \@@_renew_dots:
1488 {
1489     \cs_set_eq:NN \ldots \@@_Ldots
1490     \cs_set_eq:NN \cdots \@@_Cdots
1491     \cs_set_eq:NN \vdots \@@_Vdots
1492     \cs_set_eq:NN \ddots \@@_Ddots
1493     \cs_set_eq:NN \iddots \@@_Iddots
1494     \cs_set_eq:NN \dots \@@_Ldots
1495     \cs_set_eq:NN \hdotsfor \@@_Hdotsfor:
1496 }
1497 \cs_new_protected:Npn \@@_test_color_inside:
1498 {
1499     \bool_if:NF \l_@@_color_inside_bool
1500     {

```

We will issue an error only during the first run.

```

1501         \bool_if:NF \g_@@_aux_found_bool
1502         { \@@_error:n { without~color~inside } }
1503     }
1504 }

```

```

1505 \cs_new_protected:Npn \@@_redefine_everycr: { \everycr { \@@_everycr: } }
1506 \hook_gput_code:nnn { begindocument } { . }
1507 {
1508     \IfPackageLoadedTF { colortbl }
1509     {
1510         \cs_set_protected:Npn \@@_redefine_everycr:
1511         {
1512             \CT@everycr
1513             {
1514                 \noalign { \cs_gset_eq:NN \CT@row@color \prg_do_nothing: }
1515                 \@@_everycr:
1516             }
1517         }
1518     }
1519 }
1520 }

```

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

overwrite the previous definition of that `row` node and we have managed to avoid the error messages of that redefinition ⁴.

```

1521 \hook_gput_code:nnn { begindocument } { . }
1522 {
1523   \IfPackageLoadedTF { booktabs }
1524   {
1525     \cs_new_protected:Npn \@@_patch_booktabs:
1526     { \tl_put_left:Nn \@BTnormal \@@_create_row_node_i: }
1527   }
1528   { \cs_new_protected:Npn \@@_patch_booktabs: { } }
1529 }

```

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

```

1530 \cs_new_protected:Npn \@@_pre_array_ii:
1531 {

```

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

```

1532   \int_gzero:N \g_@@_total_X_weight_int
1533   \@@_expand_clist:N \l_@@_hlines_clist
1534   \@@_expand_clist:N \l_@@_vlines_clist
1535   \@@_patch_booktabs:
1536   \box_clear_new:N \l_@@_cell_box
1537   \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}`).

```

1538   \bool_if:NT \l_@@_small_bool
1539   {
1540     \cs_set_nopar:Npn \arraystretch { 0.47 }
1541     \dim_set:Nn \arraycolsep { 1.45 pt }

```

By default, `\@@_small_scriptstyle:` is null.

```

1542     \cs_set_eq:NN \@@_tuning_key_small: \scriptstyle
1543   }

```

```

1544   \bool_if:NT \g_@@_recreate_cell_nodes_bool
1545   {
1546     \tl_put_right:Nn \@@_begin_of_row:
1547     {
1548       \pgfsys@markposition
1549       { \@@_env: - row - \int_use:N \c@iRow - base }
1550     }
1551   }

```

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`.

```

1552   \cs_set_nopar:Npn \ialign
1553   {
1554     \@@_redefine_everycr:
1555     \tabskip = \c_zero_skip

```

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

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`.

```

1556      \dim_gzero_new:N \g_@@_dp_row_zero_dim
1557      \dim_gset:Nn \g_@@_dp_row_zero_dim { \box_dp:N \@arstrutbox }
1558      \dim_gzero_new:N \g_@@_ht_row_zero_dim
1559      \dim_gset:Nn \g_@@_ht_row_zero_dim { \box_ht:N \@arstrutbox }
1560      \dim_gzero_new:N \g_@@_ht_row_one_dim
1561      \dim_gset:Nn \g_@@_ht_row_one_dim { \box_ht:N \@arstrutbox }
1562      \dim_gzero_new:N \g_@@_dp_ante_last_row_dim
1563      \dim_gzero_new:N \g_@@_ht_last_row_dim
1564      \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1565      \dim_gzero_new:N \g_@@_dp_last_row_dim
1566      \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }

```

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

```

1567      \cs_set_eq:NN \ialign \@_old_ialign:
1568      \halign
1569  }

```

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).

```

1570      \cs_set_eq:NN \@_old_ldots \ldots
1571      \cs_set_eq:NN \@_old_cdots \cdots
1572      \cs_set_eq:NN \@_old_vdots \vdots
1573      \cs_set_eq:NN \@_old_ddots \ddots
1574      \cs_set_eq:NN \@_old_iddots \iddots
1575      \bool_if:NTF \l_@@_standard_cline_bool
1576      { \cs_set_eq:NN \cline \@_standard_cline }
1577      { \cs_set_eq:NN \cline \@_cline }
1578      \cs_set_eq:NN \Ldots \@_Ldots
1579      \cs_set_eq:NN \Cdots \@_Cdots
1580      \cs_set_eq:NN \Vdots \@_Vdots
1581      \cs_set_eq:NN \Ddots \@_Ddots
1582      \cs_set_eq:NN \Iddots \@_Iddots
1583      \cs_set_eq:NN \Hline \@_Hline:
1584      \cs_set_eq:NN \Hspace \@_Hspace:
1585      \cs_set_eq:NN \Hdotsfor \@_Hdotsfor:
1586      \cs_set_eq:NN \Vdotsfor \@_Vdotsfor:
1587      \cs_set_eq:NN \Block \@_Block:
1588      \cs_set_eq:NN \rotate \@_rotate:
1589      \cs_set_eq:NN \OnlyMainNiceMatrix \@_OnlyMainNiceMatrix:n
1590      \cs_set_eq:NN \dotfill \@_dotfill:
1591      \cs_set_eq:NN \CodeAfter \@_CodeAfter:
1592      \cs_set_eq:NN \diagbox \@_diagbox:nn
1593      \cs_set_eq:NN \NotEmpty \@_NotEmpty:
1594      \cs_set_eq:NN \RowStyle \@_RowStyle:n
1595      \seq_map_inline:Nn \l_@@_custom_line_commands_seq
1596      { \cs_set_eq:cc { ##1 } { nicematrix - ##1 } }
1597      \cs_set_eq:NN \cellcolor \@_cellcolor_tabular
1598      \cs_set_eq:NN \rowcolor \@_rowcolor_tabular
1599      \cs_set_eq:NN \rowcolors \@_rowcolors_tabular
1600      \cs_set_eq:NN \rowlistcolors \@_rowlistcolors_tabular
1601      \int_compare:nNnT \l_@@_first_row_int > \c_zero_int

```

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

```

1602     { \cs_set_eq:NN \@@_tuning_first_row: \prg_do_nothing: }
1603     \int_compare:nNt \l_@@_last_row_int < \c_zero_int
1604     { \cs_set_eq:NN \@@_tuning_last_row: \prg_do_nothing: }
1605     \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.

```

1606     \cs_set_eq:NN \multicolumn \@@_multicolumn:nnn
1607     \hook_gput_code:nnn { env / tabular / begin } { . }
1608     { \cs_set_eq:NN \multicolumn \@@_old_multicolumn }
1609     \@@_revert_colortbl:

```

If there is one or several commands `\tablarnote` 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!).

```

1610     \tl_if_exist:NT \l_@@_note_in_caption_tl
1611     {
1612         \tl_if_empty:NF \l_@@_note_in_caption_tl
1613         {
1614             \int_gset_eq:NN \g_@@_notes_caption_int \l_@@_note_in_caption_tl
1615             \int_gset:Nn \c@tablarnote { \l_@@_note_in_caption_tl }
1616         }
1617     }

```

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

```

1618     \seq_gclear:N \g_@@_multicolumn_cells_seq
1619     \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).

```

1620     \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`).

```

1621     \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.

```

1622     \int_gzero_new:N \g_@@_col_total_int
1623     \cs_set_eq:NN \@ifnextchar \new@ifnextchar
1624     \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.

```

1625     \tl_gclear_new:N \g_@@_Cdots_lines_tl
1626     \tl_gclear_new:N \g_@@_Ldots_lines_tl
1627     \tl_gclear_new:N \g_@@_Vdots_lines_tl
1628     \tl_gclear_new:N \g_@@_Ddots_lines_tl
1629     \tl_gclear_new:N \g_@@_Iddots_lines_tl
1630     \tl_gclear_new:N \g_@@_HVDotsfor_lines_tl

```

```

1631     \tl_gclear:N \g_nicematrix_code_before_tl
1632     \tl_gclear:N \g_@@_pre_code_before_tl
1633 }

```

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

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

```

1634 \cs_new_protected:Npn \@@_pre_array:
1635 {
1636   \cs_if_exist:NT \theiRow { \int_set_eq:NN \l_@@_old_iRow_int \c@iRow }
1637   \int_gzero_new:N \c@iRow
1638   \cs_if_exist:NT \thejCol { \int_set_eq:NN \l_@@_old_jCol_int \c@jCol }
1639   \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).

```

1640   \int_compare:nNnT \l_@@_last_row_int = { -1 }
1641   {
1642     \bool_set_true:N \l_@@_last_row_without_value_bool
1643     \bool_if:NT \g_@@_aux_found_bool
1644     { \int_set:Nn \l_@@_last_row_int { \seq_item:Nn \g_@@_size_seq 3 } }
1645   }
1646   \int_compare:nNnT \l_@@_last_col_int = { -1 }
1647   {
1648     \bool_if:NT \g_@@_aux_found_bool
1649     { \int_set:Nn \l_@@_last_col_int { \seq_item:Nn \g_@@_size_seq 6 } }
1650   }

```

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”.

```

1651   \int_compare:nNnT \l_@@_last_row_int > { -2 }
1652   {
1653     \tl_put_right:Nn \@@_update_for_first_and_last_row:
1654     {
1655       \dim_gset:Nn \g_@@_ht_last_row_dim
1656       { \dim_max:nn \g_@@_ht_last_row_dim { \box_ht:N \l_@@_cell_box } }
1657       \dim_gset:Nn \g_@@_dp_last_row_dim
1658       { \dim_max:nn \g_@@_dp_last_row_dim { \box_dp:N \l_@@_cell_box } }
1659     }
1660   }

1661   \seq_gclear:N \g_@@_cols_vlism_seq
1662   \seq_gclear:N \g_@@_submatrix_seq

```

Now the `\CodeBefore`.

```

1663   \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.

```

1664   \seq_gclear:N \g_@@_pos_of_blocks_seq

```

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

```

1665   \seq_gclear_new:N \g_@@_multicolumn_cells_seq
1666   \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.

```
1667 \int_gset:Nn \g_@@_last_row_node_int { -2 }
```

The value `-2` is important.

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

```
1668 \@@_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.

```
1669 \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]`).

```
1670 \dim_zero_new:N \l_@@_left_delim_dim
1671 \dim_zero_new:N \l_@@_right_delim_dim
1672 \bool_if:NTF \g_@@_delims_bool
1673 {
```

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

```
1674 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_left_delim_tl $ }
1675 \dim_set:Nn \l_@@_left_delim_dim { \box_wd:N \l_tmpa_box }
1676 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_right_delim_tl $ }
1677 \dim_set:Nn \l_@@_right_delim_dim { \box_wd:N \l_tmpa_box }
1678 }
1679 {
1680 \dim_gset:Nn \l_@@_left_delim_dim
1681 { 2 \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
1682 \dim_gset_eq:NN \l_@@_right_delim_dim \l_@@_left_delim_dim
1683 }
```

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).

```
1684 \hbox_set:Nw \l_@@_the_array_box
1685 \skip_horizontal:N \l_@@_left_margin_dim
1686 \skip_horizontal:N \l_@@_extra_left_margin_dim
1687 \c_math_toggle_token
1688 \bool_if:NTF \l_@@_light_syntax_bool
1689 { \use:c { @@-light-syntax } }
1690 { \use:c { @@-normal-syntax } }
1691 }
```

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

```
1692 \cs_new_protected_nopar:Npn \@@_CodeBefore_Body:w #1 \Body
1693 {
1694 \tl_set:Nn \l_tmpa_tl { #1 }
1695 \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
1696 { \@@_rescan_for_spanish:N \l_tmpa_tl }
1697 \tl_gput_left:N \g_@@_pre_code_before_tl \l_tmpa_tl
1698 \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...

```
1699 \@@_pre_array:
1700 }
```

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 only for legibility).

```
1701 \cs_new_protected:Npn \@@_pre_code_before:
1702 {
```

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).

```
1703 \int_set:Nn \c@iRow { \seq_item:Nn \g_@@_size_seq 2 }
1704 \int_set:Nn \c@jCol { \seq_item:Nn \g_@@_size_seq 5 }
1705 \int_set_eq:NN \g_@@_row_total_int { \seq_item:Nn \g_@@_size_seq 3 }
1706 \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.

```
1707 \pgfsys@markposition { \@@_env: - position }
1708 \pgfsys@getposition { \@@_env: - position } \@@_picture_position:
1709 \pgfpicture
1710 \pgf@relevantforpicturesizefalse
```

First, the recreation of the `row` nodes.

```
1711 \int_step_inline:nnn \l_@@_first_row_int { \g_@@_row_total_int + 1 }
1712 {
1713   \pgfsys@getposition { \@@_env: - row - ##1 } \@@_node_position:
1714   \pgfcoordinate { \@@_env: - row - ##1 }
1715   { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1716 }
```

Now, the recreation of the `col` nodes.

```
1717 \int_step_inline:nnn \l_@@_first_col_int { \g_@@_col_total_int + 1 }
1718 {
1719   \pgfsys@getposition { \@@_env: - col - ##1 } \@@_node_position:
1720   \pgfcoordinate { \@@_env: - col - ##1 }
1721   { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1722 }
```

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

```
1723 \@@_create_diag_nodes:
```

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

```
1724 \bool_if:NT \g_@@_recreate_cell_nodes_bool \@@_recreate_cell_nodes:
1725 \endpgfpicture
```

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

```
1726 \@@_create_blocks_nodes:
1727 \IfPackageLoadedTF { tikz }
1728 {
1729   \tikzset
1730   {
1731     every-picture / .style =
1732     { overlay , name-prefix = \@@_env: - }
1733   }
1734 }
1735 { }
1736 \cs_set_eq:NN \cellcolor \@@_cellcolor
1737 \cs_set_eq:NN \rectanglecolor \@@_rectanglecolor
1738 \cs_set_eq:NN \roundedrectanglecolor \@@_roundedrectanglecolor
1739 \cs_set_eq:NN \rowcolor \@@_rowcolor
```

```

1740 \cs_set_eq:NN \rowcolors \@@_rowcolors
1741 \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors
1742 \cs_set_eq:NN \arraycolor \@@_arraycolor
1743 \cs_set_eq:NN \columncolor \@@_columncolor
1744 \cs_set_eq:NN \chessboardcolors \@@_chessboardcolors
1745 \cs_set_eq:NN \SubMatrix \@@_SubMatrix_in_code_before
1746 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
1747 \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
1748 }

```

```

1749 \cs_new_protected:Npn \@@_exec_code_before:
1750 {
1751   \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`.

```

1752   \@@_add_to_colors_seq:nn { { nocolor } } { }
1753   \bool_gset_false:N \g_@@_recreate_cell_nodes_bool
1754   \group_begin:

```

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

```

1755   \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`).

```

1756   \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
1757   { \@@_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`.

```

1758   \exp_last_unbraced:NV \@@_CodeBefore_keys:
1759   \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.

```

1760   \@@_actually_color:
1761   \l_@@_code_before_tl
1762   \q_stop
1763   \bool_if:NT \l_@@_tabular_bool \c_math_toggle_token
1764   \group_end:
1765   \bool_if:NT \g_@@_recreate_cell_nodes_bool
1766   { \tl_put_left:Nn \@@_node_for_cell: \@@_patch_node_for_cell: }
1767 }

```

```

1768 \keys_define:nn { NiceMatrix / CodeBefore }
1769 {
1770   create-cell-nodes .bool_gset:N = \g_@@_recreate_cell_nodes_bool ,
1771   create-cell-nodes .default:n = true ,
1772   sub-matrix .code:n = \keys_set:nn { NiceMatrix / sub-matrix } { #1 } ,
1773   sub-matrix .value_required:n = true ,
1774   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1775   delimiters / color .value_required:n = true ,
1776   unknown .code:n = \@@_error:n { Unknown~key~for~CodeBefore }
1777 }

```

```

1778 \NewDocumentCommand \@@_CodeBefore_keys: { 0 { } }
1779 {
1780   \keys_set:nn { NiceMatrix / CodeBefore } { #1 }
1781   \@@_CodeBefore:w
1782 }

```

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.

```

1783 \cs_new_protected:Npn \@@_CodeBefore:w #1 \q_stop
1784 {
1785   \bool_if:NT \g_@@_aux_found_bool
1786   {
1787     \@@_pre_code_before:
1788     #1
1789   }
1790 }

```

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.

```

1791 \cs_new_protected:Npn \@@_recreate_cell_nodes:
1792 {
1793   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
1794   {
1795     \pgfsys@getposition { \@@_env: - ##1 - base } \@@_node_position:
1796     \pgfcoordinate { \@@_env: - row - ##1 - base }
1797     { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1798     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
1799     {
1800       \cs_if_exist:cT
1801       { pgf @ sys @ pdf @ mark @ pos @ \@@_env: - ##1 - #####1 - NW }
1802       {
1803         \pgfsys@getposition
1804         { \@@_env: - ##1 - #####1 - NW }
1805         \@@_node_position:
1806         \pgfsys@getposition
1807         { \@@_env: - ##1 - #####1 - SE }
1808         \@@_node_position_i:
1809         \@@_pgf_rect_node:nnn
1810         { \@@_env: - ##1 - #####1 }
1811         { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1812         { \pgfpointdiff \@@_picture_position: \@@_node_position_i: }
1813       }
1814     }
1815   }
1816   \int_step_inline:nn \c@iRow
1817   {
1818     \pgfnodealias
1819     { \@@_env: - ##1 - last }
1820     { \@@_env: - ##1 - \int_use:N \c@jCol }
1821   }
1822   \int_step_inline:nn \c@jCol
1823   {
1824     \pgfnodealias
1825     { \@@_env: - last - ##1 }
1826     { \@@_env: - \int_use:N \c@iRow - ##1 }
1827   }
1828   \@@_create_extra_nodes:
1829 }

```

```

1830 \cs_new_protected:Npn \@@_create_blocks_nodes:
1831 {
1832   \pgfpicture
1833   \pgf@relevantforpicturesizefalse
1834   \pgfrememberpicturepositiononpagetrue
1835   \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
1836   { \@@_create_one_block_node:nnnnn ##1 }
1837   \endpgfpicture
1838 }

```

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

```

1839 \cs_new_protected:Npn \@@_create_one_block_node:nnnnn #1 #2 #3 #4 #5
1840 {
1841   \tl_if_empty:nF { #5 }
1842   {
1843     \@@_qpoint:n { col - #2 }
1844     \dim_set_eq:NN \l_tmpa_dim \pgf@x
1845     \@@_qpoint:n { #1 }
1846     \dim_set_eq:NN \l_tmpb_dim \pgf@y
1847     \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
1848     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
1849     \@@_qpoint:n { \int_eval:n { #3 + 1 } }
1850     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
1851     \@@_pgf_rect_node:nnnnn
1852     { \@@_env: - #5 }
1853     { \dim_use:N \l_tmpa_dim }
1854     { \dim_use:N \l_tmpb_dim }
1855     { \dim_use:N \l_@@_tmpc_dim }
1856     { \dim_use:N \l_@@_tmpd_dim }
1857   }
1858 }

```

```

1859 \cs_new_protected:Npn \@@_patch_for_revtext:
1860 {
1861   \cs_set_eq:NN \@addamp \@addamp@LaTeX
1862   \cs_set_eq:NN \insert@column \insert@column@array
1863   \cs_set_eq:NN \@classx \@classx@array
1864   \cs_set_eq:NN \@xarraycr \@xarraycr@array
1865   \cs_set_eq:NN \@arraycr \@arraycr@array
1866   \cs_set_eq:NN \@xargarraycr \@xargarraycr@array
1867   \cs_set_eq:NN \array \array@array
1868   \cs_set_eq:NN \@array \@array@array
1869   \cs_set_eq:NN \@tabular \@tabular@array
1870   \cs_set_eq:NN \@mkpream \@mkpream@array
1871   \cs_set_eq:NN \endarray \endarray@array
1872   \cs_set:Npn \@tabarray { \ifnextchar [ { \@array } { \@array [ c ] } }
1873   \cs_set:Npn \endtabular { \endarray $\egroup} % $
1874 }

```

11 The environment `{NiceArrayWithDelims}`

```

1875 \NewDocumentEnvironment { NiceArrayWithDelims }
1876 { m m 0 { } m ! 0 { } t \CodeBefore }
1877 {

```

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).

```

1878 \bool_if:NT \c_@@_revtex_bool \@@_patch_for_revtex:
1879 \@@_provide_pgfsyspdfmark:
1880 \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.

```

1881 \bgroup

1882 \tl_gset:Nn \g_@@_left_delim_tl { #1 }
1883 \tl_gset:Nn \g_@@_right_delim_tl { #2 }
1884 \tl_gset:Nn \g_@@_user_preamble_tl { #4 }

1885 \int_gzero:N \g_@@_block_box_int
1886 \dim_zero:N \g_@@_width_last_col_dim
1887 \dim_zero:N \g_@@_width_first_col_dim
1888 \bool_gset_false:N \g_@@_row_of_col_done_bool
1889 \str_if_empty:NT \g_@@_name_env_str
1890 { \str_gset:Nn \g_@@_name_env_str { NiceArrayWithDelims } }
1891 \bool_if:NTF \l_@@_tabular_bool
1892 \mode_leave_vertical:
1893 \@@_test_if_math_mode:
1894 \bool_if:NT \l_@@_in_env_bool { \@@_fatal:n { Yet~in~env } }
1895 \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.

```

1896 \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.

```

1897 \cs_if_exist:NT \tikz@library@external@loaded
1898 {
1899 \tikzexternaldisable
1900 \cs_if_exist:NT \ifstandalone
1901 { \tikzset { external / optimize = false } }
1902 }

```

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

```

1903 \int_gincr:N \g_@@_env_int
1904 \bool_if:NF \l_@@_block_auto_columns_width_bool
1905 { \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`).

```

1906 \seq_gclear:N \g_@@_blocks_seq
1907 \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`.

```

1908 \seq_gclear:N \g_@@_pos_of_stroken_blocks_seq
1909 \seq_gclear:N \g_@@_pos_of_xdots_seq
1910 \tl_gclear_new:N \g_@@_code_before_tl
1911 \tl_gclear:N \g_@@_row_style_tl

```

We load all the informations written in the aux file during previous compilations corresponding to the current environment.

⁷e.g. `\color[rgb]{0.5,0.5,0}`

```

1912 \tl_if_exist:cTF { c_@@ _ \int_use:N \g_@@_env_int _ tl }
1913 {
1914   \bool_gset_true:N \g_@@_aux_found_bool
1915   \use:c { c_@@ _ \int_use:N \g_@@_env_int _ tl }
1916 }
1917 { \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.

```

1918 \tl_gclear:N \g_@@_aux_tl
1919 \tl_if_empty:NF \g_@@_code_before_tl
1920 {
1921   \bool_set_true:N \l_@@_code_before_bool
1922   \tl_put_right:NV \l_@@_code_before_tl \g_@@_code_before_tl
1923 }
1924 \tl_if_empty:NF \g_@@_pre_code_before_tl
1925 { \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.

```

1926 \bool_if:NTF \g_@@_delims_bool
1927 { \keys_set:nn { NiceMatrix / pNiceArray } }
1928 { \keys_set:nn { NiceMatrix / NiceArray } }
1929 { #3 , #5 }

1930 \@@_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:.

```

1931 \IfBooleanTF { #6 } \@@_CodeBefore_Body:w \@@_pre_array:
1932 }

```

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

```

1933 {
1934   \bool_if:NTF \l_@@_light_syntax_bool
1935   { \use:c { end @@-light-syntax } }
1936   { \use:c { end @@-normal-syntax } }
1937   \c_math_toggle_token
1938   \skip_horizontal:N \l_@@_right_margin_dim
1939   \skip_horizontal:N \l_@@_extra_right_margin_dim
1940   \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.

```

1941 \bool_if:NT \l_@@_width_used_bool
1942 {
1943   \int_if_zero:nT \g_@@_total_X_weight_int
1944   { \@@_error_or_warning:n { width-without-X-columns } }
1945 }

```

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.

```

1946 \int_compare:nNnT \g_@@_total_X_weight_int > \c_zero_int
1947 {
1948   \tl_gput_right:Nx \g_@@_aux_tl
1949   {

```

```

1950      \bool_set_true:N \l_@@_X_columns_aux_bool
1951      \dim_set:Nn \l_@@_X_columns_dim
1952      {
1953        \dim_compare:nNnTF
1954        {
1955          \dim_abs:n
1956          { \l_@@_width_dim - \box_wd:N \l_@@_the_array_box }
1957        }
1958        <
1959        { 0.001 pt }
1960        { \dim_use:N \l_@@_X_columns_dim }
1961        {
1962          \dim_eval:n
1963          {
1964            ( \l_@@_width_dim - \box_wd:N \l_@@_the_array_box )
1965            / \int_use:N \g_@@_total_X_weight_int
1966            + \l_@@_X_columns_dim
1967          }
1968        }
1969      }
1970    }
1971  }

```

It 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).

```

1972    \int_compare:nNnT \l_@@_last_row_int > { -2 }
1973    {
1974      \bool_if:NF \l_@@_last_row_without_value_bool
1975      {
1976        \int_compare:nNnF \l_@@_last_row_int = \c@iRow
1977        {
1978          \@@_error:n { Wrong-last-row }
1979          \int_gset_eq:NN \l_@@_last_row_int \c@iRow
1980        }
1981      }
1982    }

```

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

```

1983    \int_gset_eq:NN \c@jCol \g_@@_col_total_int
1984    \bool_if:NnTF \g_@@_last_col_found_bool
1985    { \int_gdecr:N \c@jCol }
1986    {
1987      \int_compare:nNnT \l_@@_last_col_int > { -1 }
1988      { \@@_error:n { last-col-not-used } }
1989    }

```

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

```

1990    \int_gset_eq:NN \g_@@_row_total_int \c@iRow
1991    \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. 88).

```

1992    \int_if_zero:nT \l_@@_first_col_int
1993    { \skip_horizontal:N \g_@@_width_first_col_dim }

```

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

```

1994    \bool_if:nTF { ! \g_@@_delims_bool }
1995    {

```

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

```

1996 \tl_if_eq:NNTF \l_@@_baseline_tl \c_@@_c_tl
1997 \@@_use_arraybox_with_notes_c:
1998 {
1999 \tl_if_eq:NNTF \l_@@_baseline_tl \c_@@_b_tl
2000 \@@_use_arraybox_with_notes_b:
2001 \@@_use_arraybox_with_notes:
2002 }
2003 }

```

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).

```

2004 {
2005 \int_if_zero:nTF \l_@@_first_row_int
2006 {
2007 \dim_set_eq:NN \l_tmpa_dim \g_@@_dp_row_zero_dim
2008 \dim_add:Nn \l_tmpa_dim \g_@@_ht_row_zero_dim
2009 }
2010 { \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

```

2011 \int_compare:nNnTF \l_@@_last_row_int > { -2 }
2012 {
2013 \dim_set_eq:NN \l_tmpb_dim \g_@@_ht_last_row_dim
2014 \dim_add:Nn \l_tmpb_dim \g_@@_dp_last_row_dim
2015 }
2016 { \dim_zero:N \l_tmpb_dim }
2017 \hbox_set:Nn \l_tmpa_box
2018 {
2019 \c_math_toggle_token
2020 \@@_color:o \l_@@_delimiters_color_tl
2021 \exp_after:wN \left \g_@@_left_delim_tl
2022 \vcenter
2023 {

```

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.

```

2024 \skip_vertical:n { -\l_tmpa_dim - \arrayrulewidth }
2025 \hbox
2026 {
2027 \bool_if:NNTF \l_@@_tabular_bool
2028 { \skip_horizontal:N -\tabcolsep }
2029 { \skip_horizontal:N -\arraycolsep }
2030 \@@_use_arraybox_with_notes_c:
2031 \bool_if:NNTF \l_@@_tabular_bool
2032 { \skip_horizontal:N -\tabcolsep }
2033 { \skip_horizontal:N -\arraycolsep }
2034 }

```

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

```

2035 \skip_vertical:N -\l_tmpb_dim
2036 \skip_vertical:N \arrayrulewidth
2037 }
2038 \exp_after:wN \right \g_@@_right_delim_tl
2039 \c_math_toggle_token
2040 }

```

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.

```

2041 \bool_if:NNTF \l_@@_delimiters_max_width_bool
2042 {

```

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).

```

2043         \@@_put_box_in_flow_bis:nn
2044         \g_@@_left_delim_tl
2045         \g_@@_right_delim_tl
2046     }
2047     \@@_put_box_in_flow:
2048 }

```

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. 89).

```

2049 \bool_if:NT \g_@@_last_col_found_bool
2050 { \skip_horizontal:N \g_@@_width_last_col_dim }
2051 \bool_if:NT \l_@@_preamble_bool
2052 {
2053     \int_compare:nNtT \c@jCol < \g_@@_static_num_of_col_int
2054     { \@@_warning_gredirect_none:n { columns~not~used } }
2055 }
2056 \@@_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.

```

2057 \egroup

```

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

```

2058 \iow_now:Nn \@mainaux { \ExplSyntaxOn }
2059 \iow_now:Nn \@mainaux { \char_set_catcode_space:n { 32 } }
2060 \iow_now:Nx \@mainaux
2061 {
2062     \tl_gset:cn { c_@@_ \int_use:N \g_@@_env_int _ tl }
2063     { \exp_not:o \g_@@_aux_tl }
2064 }
2065 \iow_now:Nn \@mainaux { \ExplSyntaxOff }

2066 \bool_if:NT \g_@@_footnote_bool \endsavenotes
2067 }

```

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 that 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.

```

2068 \cs_new_protected:Npn \@@_transform_preamble:
2069 {
2070     \@@_transform_preamble_i:
2071     \@@_transform_preamble_ii:
2072 }

2073 \cs_new_protected:Npn \@@_transform_preamble_i:
2074 {
2075     \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`).

```

2076 \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.

```

2077 \bool_gset_false:N \g_tmpb_bool

```

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

```
2078 \tl_gclear_new:N \g_@@_pre_cell_tl
```

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

```
2079 \int_zero:N \l_tmpa_int
2080 \tl_gclear:N \g_@@_array_preamble_tl
2081 \tl_if_eq:NNTF \l_@@_vlines_clist \c_@@_all_tl
2082 {
2083   \tl_gset:Nn \g_@@_array_preamble_tl
2084     { ! { \skip_horizontal:N \arrayrulewidth } }
2085 }
2086 {
2087   \clist_if_in:NnT \l_@@_vlines_clist 1
2088   {
2089     \tl_gset:Nn \g_@@_array_preamble_tl
2090       { ! { \skip_horizontal:N \arrayrulewidth } }
2091   }
2092 }
```

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

```
2093 \exp_last_unbraced:NV \@@_rec_preamble:n \g_@@_user_preamble_tl \stop
2094 \int_gset_eq:NN \g_@@_static_num_of_col_int \c@jCol
```

```
2095 \@@_replace_columncolor:
2096 }
```

```
2097 \hook_gput_code:nnn { begindocument } { . }
2098 {
2099   \IfPackageLoadedTF { colortbl }
2100   {
2101     \regex_const:Nn \c_@@_columncolor_regex { \c { columncolor } }
2102     \cs_new_protected:Npn \@@_replace_columncolor:
2103       {
2104         \regex_replace_all:NnN
2105           \c_@@_columncolor_regex
2106           { \c { @@_columncolor_preamble } }
2107           \g_@@_array_preamble_tl
2108       }
2109   }
2110   {
2111     \cs_new_protected:Npn \@@_replace_columncolor:
2112       { \cs_set_eq:NN \columncolor \@@_columncolor_preamble }
2113   }
2114 }
```

```
2115 \cs_new_protected:Npn \@@_transform_preamble_ii:
2116 {
```

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

```
2117 \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2118 {
2119   \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2120   { \bool_gset_true:N \g_@@_delims_bool }
2121 }
2122 { \bool_gset_true:N \g_@@_delims_bool }
```

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

```
2123 \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).

```

2124 \int_if_zero:nTF \l_@@_first_col_int
2125 { \tl_gput_left:No \g_@@_array_preamble_tl \c_@@_preamble_first_col_tl }
2126 {
2127   \bool_if:NF \g_@@_delims_bool
2128   {
2129     \bool_if:NF \l_@@_tabular_bool
2130     {
2131       \tl_if_empty:NT \l_@@_vlines_clist
2132       {
2133         \bool_if:NF \l_@@_exterior_arraycolsep_bool
2134         { \tl_gput_left:Nn \g_@@_array_preamble_tl { @ { } } }
2135       }
2136     }
2137   }
2138 }
2139 \int_compare:nNnTF \l_@@_last_col_int > { -1 }
2140 { \tl_gput_right:No \g_@@_array_preamble_tl \c_@@_preamble_last_col_tl }
2141 {
2142   \bool_if:NF \g_@@_delims_bool
2143   {
2144     \bool_if:NF \l_@@_tabular_bool
2145     {
2146       \tl_if_empty:NT \l_@@_vlines_clist
2147       {
2148         \bool_if:NF \l_@@_exterior_arraycolsep_bool
2149         { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { } } }
2150       }
2151     }
2152   }
2153 }

```

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`).

```

2154 \dim_compare:nNnT \l_@@_tabular_width_dim = \c_zero_dim
2155 {
2156   \tl_gput_right:Nn \g_@@_array_preamble_tl
2157   { > { \@@_error_too_much_cols: } 1 }
2158 }
2159 }

```

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.

```

2160 \cs_new_protected:Npn \@@_rec_preamble:n #1
2161 {

```

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

```

2162 \cs_if_exist:cTF { @@ _ \token_to_str:N #1 }
2163 { \use:c { @@ _ \token_to_str:N #1 } { #1 } }
2164 {

```

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

```

2165 \cs_if_exist:cTF { NC @ find @ #1 }
2166 {
2167   \tl_set_eq:Nc \l_tmpb_tl { NC @ rewrite @ #1 }

```

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`.

```

2168         \exp_last_unbraced:NV \@@_rec_preamble:n \l_tmpb_tl
2169     }
2170     {
2171         \tl_if_eq:nnT { #1 } { S }
2172         { \@@_fatal:n { unknown~column~type~S } }
2173         { \@@_fatal:nn { unknown~column~type } { #1 } }
2174     }
2175 }
2176 }

```

For c, l and r

```

2177 \cs_new:Npn \@@_c #1
2178 {
2179     \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2180     \tl_gclear:N \g_@@_pre_cell_tl
2181     \tl_gput_right:Nn \g_@@_array_preamble_tl
2182     { > \@@_cell_begin:w c < \@@_cell_end: }

```

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

```

2183     \int_gincr:N \c@jCol
2184     \@@_rec_preamble_after_col:n
2185 }

2186 \cs_new:Npn \@@_l #1
2187 {
2188     \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2189     \tl_gclear:N \g_@@_pre_cell_tl
2190     \tl_gput_right:Nn \g_@@_array_preamble_tl
2191     {
2192         > { \@@_cell_begin:w \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_l_tl }
2193         l
2194         < \@@_cell_end:
2195     }
2196     \int_gincr:N \c@jCol
2197     \@@_rec_preamble_after_col:n
2198 }

2199 \cs_new:Npn \@@_r #1
2200 {
2201     \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2202     \tl_gclear:N \g_@@_pre_cell_tl
2203     \tl_gput_right:Nn \g_@@_array_preamble_tl
2204     {
2205         > { \@@_cell_begin:w \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_r_tl }
2206         r
2207         < \@@_cell_end:
2208     }
2209     \int_gincr:N \c@jCol
2210     \@@_rec_preamble_after_col:n
2211 }

```

For ! and @

```

2212 \cs_new:cpn { @@ _ \token_to_str:N ! } #1 #2
2213 {
2214     \tl_gput_right:Nn \g_@@_array_preamble_tl { #1 { #2 } }
2215     \@@_rec_preamble:n
2216 }
2217 \cs_set_eq:cc { @@ _ \token_to_str:N @ } { @@ _ \token_to_str:N ! }

```

For |

```

2218 \cs_new:cpn { @@ _ | } #1
2219 {

```

`\l_tmpa_int` is the number of successive occurrences of |

```

2220 \int_incr:N \l_tmpa_int
2221 \@@_make_preamble_i_i:n
2222 }

2223 \cs_new_protected:Npn \@@_make_preamble_i_i:n #1
2224 {
2225   \str_if_eq:nnTF { #1 } |
2226   { \use:c { @@ _ | } | }
2227   { \@@_make_preamble_i_ii:nn { } #1 }
2228 }

2229 \cs_new_protected:Npn \@@_make_preamble_i_ii:nn #1 #2
2230 {
2231   \str_if_eq:nnTF { #2 } [
2232   { \@@_make_preamble_i_ii:nw { #1 } [ ] }
2233   { \@@_make_preamble_i_iii:nn { #2 } { #1 } }
2234 }
2235 \cs_new_protected:Npn \@@_make_preamble_i_ii:nw #1 [ #2 ]
2236 { \@@_make_preamble_i_ii:nn { #1 , #2 } }

2237 \cs_new_protected:Npn \@@_make_preamble_i_iii:nn #1 #2
2238 {
2239   \@@_compute_rule_width:n { multiplicity = \l_tmpa_int , #2 }
2240   \tl_gput_right:Nx \g_@@_array_preamble_tl
2241   {

```

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

```

2242   \exp_not:N ! { \skip_horizontal:n { \dim_eval:n { \l_@@_rule_width_dim } } }
2243   }
2244   \tl_gput_right:Nx \g_@@_pre_code_after_tl
2245   {
2246     \@@_vline:n
2247     {
2248       position = \int_eval:n { \c@jCol + 1 } ,
2249       multiplicity = \int_use:N \l_tmpa_int ,
2250       total-width = \dim_use:N \l_@@_rule_width_dim ,
2251       #2
2252     }

```

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.

```

2253   }
2254   \int_zero:N \l_tmpa_int
2255   \str_if_eq:nnT { #1 } { \stop } { \bool_gset_true:N \g_tmpb_bool }
2256   \@@_rec_preamble:n #1
2257   }

2258 \cs_new:cpn { @@ _ > } #1 #2
2259 {
2260   \tl_gput_right:Nn \g_@@_pre_cell_tl { > { #2 } }
2261   \@@_rec_preamble:n
2262   }

2263 \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.

```

2264 \keys_define:nn { WithArrows / p-column }
2265 {
2266   r .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_r_str ,
2267   r .value_forbidden:n = true ,
2268   c .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_c_str ,
2269   c .value_forbidden:n = true ,

```

```

2270 l .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_l_str ,
2271 l .value_forbidden:n = true ,
2272 R .code:n =
2273   \IfPackageLoadedTF { ragged2e }
2274   { \str_set_eq:NN \l_@@_hpos_col_str \c_@@_R_str }
2275   {
2276     \@@_error_or_warning:n { ragged2e~not~loaded }
2277     \str_set_eq:NN \l_@@_hpos_col_str \c_@@_r_str
2278   } ,
2279 R .value_forbidden:n = true ,
2280 L .code:n =
2281   \IfPackageLoadedTF { ragged2e }
2282   { \str_set_eq:NN \l_@@_hpos_col_str \c_@@_L_str }
2283   {
2284     \@@_error_or_warning:n { ragged2e~not~loaded }
2285     \str_set_eq:NN \l_@@_hpos_col_str \c_@@_l_str
2286   } ,
2287 L .value_forbidden:n = true ,
2288 C .code:n =
2289   \IfPackageLoadedTF { ragged2e }
2290   { \str_set_eq:NN \l_@@_hpos_col_str \c_@@_C_str }
2291   {
2292     \@@_error_or_warning:n { ragged2e~not~loaded }
2293     \str_set_eq:NN \l_@@_hpos_col_str \c_@@_c_str
2294   } ,
2295 C .value_forbidden:n = true ,
2296 S .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_si_str ,
2297 S .value_forbidden:n = true ,
2298 p .code:n = \str_set:Nn \l_@@_vpos_col_str { p } ,
2299 p .value_forbidden:n = true ,
2300 t .meta:n = p ,
2301 m .code:n = \str_set:Nn \l_@@_vpos_col_str { m } ,
2302 m .value_forbidden:n = true ,
2303 b .code:n = \str_set:Nn \l_@@_vpos_col_str { b } ,
2304 b .value_forbidden:n = true ,
2305 }

```

For p, b and m.

```

2306 \cs_new:Npn \@@_p #1
2307 {
2308   \str_set:Nn \l_@@_vpos_col_str { #1 }
2309   \@@_make_preamble_ii_i:n
2310 }
2311 \cs_set_eq:NN \@@_b \@@_p
2312 \cs_set_eq:NN \@@_m \@@_p
2313 \cs_new_protected:Npn \@@_make_preamble_ii_i:n #1
2314 {
2315   \str_if_eq:nnTF { #1 } { [ ] }
2316   { \@@_make_preamble_ii_ii:w [ ] }
2317   { \@@_make_preamble_ii_ii:w [ ] { #1 } }
2318 }
2319 \cs_new_protected:Npn \@@_make_preamble_ii_ii:w [ #1 ]
2320 { \@@_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.

```

2321 \cs_new_protected:Npn \@@_make_preamble_ii_iii:nn #1 #2
2322 {

```

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).

```

2323 \str_set_eq:NN \l_@@_hpos_col_str \c_@@_j_str
2324 \@@_keys_p_column:n { #1 }
2325 \@@_make_preamble_ii_iv:nnn { #2 } { minipage } { }
2326 }

2327 \cs_new_protected:Npn \@@_keys_p_column:n #1
2328 { \keys_set_known:nnN { WithArrows / 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.

```

2329 \cs_new_protected:Npn \@@_make_preamble_ii_iv:nnn #1 #2 #3
2330 {
2331   \use:e
2332   {
2333     \@@_make_preamble_ii_v:nnnnnnnn
2334     { \str_if_eq:onTF \l_@@_vpos_col_str { p } { t } { b } }
2335     { \dim_eval:n { #1 } }
2336     {

```

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.

```

2337 \str_if_eq:NNTF \l_@@_hpos_col_str \c_@@_j_str
2338 { \tl_clear:N \exp_not:N \l_@@_hpos_cell_tl }
2339 {
2340   \cs_set_nopar:Npn \exp_not:N \l_@@_hpos_cell_tl
2341   { \str_lowercase:V \l_@@_hpos_col_str }
2342 }
2343 \str_case:on \l_@@_hpos_col_str
2344 {
2345   c { \exp_not:N \centering }
2346   l { \exp_not:N \raggedright }
2347   r { \exp_not:N \raggedleft }
2348   C { \exp_not:N \Centering }
2349   L { \exp_not:N \RaggedRight }
2350   R { \exp_not:N \RaggedLeft }
2351 }
2352 #3
2353 }
2354 { \str_if_eq:onT \l_@@_vpos_col_str { m } \@@_center_cell_box: }
2355 { \str_if_eq:onT \l_@@_hpos_col_str { si } \siunitx_cell_begin:w }
2356 { \str_if_eq:onT \l_@@_hpos_col_str { si } \siunitx_cell_end: }
2357 { #2 }
2358 {
2359   \str_case:onF \l_@@_hpos_col_str
2360   {
2361     { j } { c }
2362     { si } { c }
2363   }

```

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

```

2364 { \str_lowercase:V \l_@@_hpos_col_str }
2365 }
2366 }

```

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

```

2367 \int_gincr:N \c@jCol
2368 \@@_rec_preamble_after_col:n
2369 }

```

#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*.

```

2370 \cs_new_protected:Npn \@@_make_preamble_ii_v:nnnnnnnn #1 #2 #3 #4 #5 #6 #7 #8
2371 {
2372   \tl_if_eq:NNTF \l_@@_hpos_col_str \c_@@_si_str
2373     { \tl_gput_right:Nn \g_@@_array_preamble_tl { > { \@@_test_if_empty_for_S: } } }
2374     { \tl_gput_right:Nn \g_@@_array_preamble_tl { > { \@@_test_if_empty: } } }
2375   \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2376   \tl_gclear:N \g_@@_pre_cell_tl
2377   \tl_gput_right:Nn \g_@@_array_preamble_tl
2378     {
2379       > {

```

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.

```

2380       \dim_set:Nn \l_@@_col_width_dim { #2 }
2381       \@@_cell_begin:w

```

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

```

2382       \use:c { #7 } [ #1 ] { #2 }

```

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

```

2383       \everypar
2384       {
2385         \vrule height \box_ht:N \@arstrutbox width \c_zero_dim
2386         \everypar { }
2387       }

```

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

```

2388       #3

```

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

```

2389       \g_@@_row_style_tl
2390       \arraybackslash
2391       #5
2392     }
2393     #8
2394     < {
2395       #6

```

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

```

2396       \@finalstrut \@arstrutbox
2397       \use:c { end #7 }

```

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

```

2398       #4
2399       \@@_cell_end:
2400     }
2401   }
2402 }

```

```

2403 \str_new:N \c_@@_ignorespaces_str
2404 \str_set:Nx \c_@@_ignorespaces_str { \ignorespaces }
2405 \str_remove_all:Nn \c_@@_ignorespaces_str { ~ }

```

In order to test whether a cell is empty, we test whether it begins by `\ignorespaces\unskip`. However, in some circumstances, for example when `\collectcell` of `collcell` is used, the cell does not begin with `\ignorespaces`. In that case, we consider as not empty...

First, we test if the next token is `\ignorespaces` and it's not very easy...

```

2406 \cs_new_protected:Npn \@@_test_if_empty: { \peek_after:Nw \@@_test_if_empty_i: }
2407 \cs_new_protected:Npn \@@_test_if_empty_i:
2408 {
2409   \str_set:Nx \l_tmpa_str { \token_to_meaning:N \l_peek_token }
2410   \str_if_eq:NNT \l_tmpa_str \c_@@_ignorespaces_str
2411   { \@@_test_if_empty:w }
2412 }
2413 \cs_new_protected:Npn \@@_test_if_empty:w \ignorespaces
2414 {
2415   \peek_meaning:NT \unskip
2416   {
2417     \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2418     {
2419       \box_set_wd:Nn \l_@@_cell_box \c_zero_dim
2420       \skip_horizontal:N \l_@@_col_width_dim
2421     }
2422   }
2423 }
2424 \cs_new_protected:Npn \@@_test_if_empty_for_S:
2425 {
2426   \peek_meaning:NT \__siunitx_table_skip:n
2427   {
2428     \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2429     { \box_set_wd:Nn \l_@@_cell_box \c_zero_dim }
2430   }
2431 }

```

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 `\@arstrutbox`, there is only one row.

```

2432 \cs_new_protected:Npn \@@_center_cell_box:
2433 {

```

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

```

2434   \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2435   {
2436     \int_compare:nNnT
2437     { \box_ht:N \l_@@_cell_box }
2438     >

```

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).

```

2439     { \box_ht:N \strutbox }
2440     {
2441       \hbox_set:Nn \l_@@_cell_box
2442       {
2443         \box_move_down:nn
2444         {
2445           ( \box_ht:N \l_@@_cell_box - \box_ht:N \@arstrutbox

```

```

2446             + \baselineskip ) / 2
2447         }
2448         { \box_use:N \l_@@_cell_box }
2449     }
2450 }
2451 }
2452 }

```

For V (similar to the V of varwidth).

```

2453 \cs_new:Npn \@@_V #1 #2
2454 {
2455     \str_if_eq:nnTF { #2 } { [ ] }
2456     { \@@_make_preamble_V_i:w [ ] }
2457     { \@@_make_preamble_V_i:w [ ] { #2 } }
2458 }
2459 \cs_new_protected:Npn \@@_make_preamble_V_i:w [ #1 ]
2460 { \@@_make_preamble_V_ii:nn { #1 } }
2461 \cs_new_protected:Npn \@@_make_preamble_V_ii:nn #1 #2
2462 {
2463     \str_set:Nn \l_@@_vpos_col_str { p }
2464     \str_set_eq:NN \l_@@_hpos_col_str \c_@@_j_str
2465     \@@_keys_p_column:n { #1 }
2466     \IfPackageLoadedTF { varwidth }
2467     { \@@_make_preamble_ii_iv:nnn { #2 } { varwidth } { } }
2468     {
2469         \@@_error_or_warning:n { varwidth-not-loaded }
2470         \@@_make_preamble_ii_iv:nnn { #2 } { minipage } { }
2471     }
2472 }

```

For w and W

```

2473 \cs_new:Npn \@@_w { \@@_make_preamble_w:nnnn { } }
2474 \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.

```

2475 \cs_new_protected:Npn \@@_make_preamble_w:nnnn #1 #2 #3 #4
2476 {
2477     \str_if_eq:nnTF { #3 } { s }
2478     { \@@_make_preamble_w_i:nnnn { #1 } { #4 } }
2479     { \@@_make_preamble_w_ii:nnnn { #1 } { #2 } { #3 } { #4 } }
2480 }

```

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.

```

2481 \cs_new_protected:Npn \@@_make_preamble_w_i:nnnn #1 #2
2482 {
2483     \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2484     \tl_gclear:N \g_@@_pre_cell_tl
2485     \tl_gput_right:Nn \g_@@_array_preamble_tl
2486     {
2487         > {
2488             \dim_set:Nn \l_@@_col_width_dim { #2 }
2489             \@@_cell_begin:w
2490             \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_c_tl
2491         }
2492         c
2493         < {
2494             \@@_cell_end_for_w_s:

```

```

2495         #1
2496         \@@_adjust_size_box:
2497         \box_use_drop:N \l_@@_cell_box
2498     }
2499 }
2500 \int_gincr:N \c@jCol
2501 \@@_rec_preamble_after_col:n
2502 }

```

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

```

2503 \cs_new_protected:Npn \@@_make_preamble_w_ii:nnnn #1 #2 #3 #4
2504 {
2505     \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2506     \tl_gclear:N \g_@@_pre_cell_tl
2507     \tl_gput_right:Nn \g_@@_array_preamble_tl
2508     {
2509         > {

```

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.

```

2510         \dim_set:Nn \l_@@_col_width_dim { #4 }
2511         \hbox_set:Nw \l_@@_cell_box
2512         \@@_cell_begin:w
2513         \cs_set_nopar:Npn \l_@@_hpos_cell_tl { #3 }
2514     }
2515     c
2516     < {
2517         \@@_cell_end:
2518         \hbox_set_end:
2519         #1
2520         \@@_adjust_size_box:
2521         \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2522     }
2523 }

```

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

```

2524     \int_gincr:N \c@jCol
2525     \@@_rec_preamble_after_col:n
2526 }

```

```

2527 \cs_new_protected:Npn \@@_special_W:
2528 {
2529     \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \l_@@_col_width_dim
2530     { \@@_warning:n { W~warning } }
2531 }

```

For S (of siunitx).

```

2532 \cs_new:Npn \@@_S #1 #2
2533 {
2534     \str_if_eq:nnTF { #2 } { [ ] }
2535     { \@@_make_preamble_S:w [ ] }
2536     { \@@_make_preamble_S:w [ ] { #2 } }
2537 }
2538 \cs_new_protected:Npn \@@_make_preamble_S:w [ #1 ]
2539 { \@@_make_preamble_S:i:n { #1 } }
2540 \cs_new_protected:Npn \@@_make_preamble_S:i:n #1
2541 {
2542     \IfPackageLoadedTF { siunitx }
2543     {
2544         \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2545         \tl_gclear:N \g_@@_pre_cell_tl
2546         \tl_gput_right:Nn \g_@@_array_preamble_tl

```

```

2547     {
2548     > {
2549         \@@_cell_begin:w
2550         \keys_set:nn { siunitx } { #1 }
2551         \siunitx_cell_begin:w
2552     }
2553     c
2554     < { \siunitx_cell_end: \@@_cell_end: }
2555 }

```

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

```

2556     \int_gincr:N \c@jCol
2557     \@@_rec_preamble_after_col:n
2558 }
2559 { \@@_fatal:n { siunitx~not~loaded } }
2560 }

```

For (, [and \{.

```

2561 \cs_new:cpn { @@ _ \token_to_str:N ( } #1 #2
2562 {
2563     \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.

```

2564     \int_if_zero:nTF \c@jCol
2565     {
2566         \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2567         {

```

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

```

2568             \tl_gset:Nn \g_@@_left_delim_tl { #1 }
2569             \tl_gset_eq:NN \g_@@_right_delim_tl \c_@@_dot_tl
2570             \@@_rec_preamble:n #2
2571         }
2572         {
2573             \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2574             \@@_make_preamble_iv:nn { #1 } { #2 }
2575         }
2576     }
2577     { \@@_make_preamble_iv:nn { #1 } { #2 } }
2578 }
2579 \cs_set_eq:cc { @@ _ \token_to_str:N [ } { @@ _ \token_to_str:N ( }
2580 \cs_set_eq:cc { @@ _ \token_to_str:N \{ } { @@ _ \token_to_str:N ( }
2581 \cs_new_protected:Npn \@@_make_preamble_iv:nn #1 #2
2582 {
2583     \tl_gput_right:Nx \g_@@_pre_code_after_tl
2584     { \@@_delimiter:nnn #1 { \int_eval:n { \c@jCol + 1 } } \c_true_bool }
2585     \tl_if_in:nnTF { ( [ \{ ) ] \} \left \right } { #2 }
2586     {
2587         \@@_error:nn { delimiter~after~opening } { #2 }
2588         \@@_rec_preamble:n
2589     }
2590     { \@@_rec_preamble:n #2 }
2591 }

```

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

```

2592 \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}).

```

2593 \cs_new:cpn { @@ _ \token_to_str:N ) } #1 #2

```

```

2594 {
2595   \bool_if:NT \l_@@_small_bool { \@@_fatal:n { Delimiter~with~small } }
2596   \tl_if_in:nnTF { ) ] \} } { #2 }
2597   { \@@_make_preamble_v:nnn #1 #2 }
2598   {
2599     \tl_if_eq:nnTF { \stop } { #2 }
2600     {
2601       \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2602       { \tl_gset:Nn \g_@@_right_delim_tl { #1 } }
2603       {
2604         \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2605         \tl_gput_right:Nx \g_@@_pre_code_after_tl
2606         { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2607         \@@_rec_preamble:n #2
2608       }
2609     }
2610     {
2611       \tl_if_in:nnT { ( [ \{ \left } { #2 }
2612       { \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } } }
2613       \tl_gput_right:Nx \g_@@_pre_code_after_tl
2614       { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2615       \@@_rec_preamble:n #2
2616     }
2617   }
2618 }
2619 \cs_set_eq:cc { @@ _ \token_to_str:N ] } { @@ _ \token_to_str:N ] }
2620 \cs_set_eq:cc { @@ _ \token_to_str:N \} } { @@ _ \token_to_str:N \} }
2621 \cs_new_protected:Npn \@@_make_preamble_v:nnn #1 #2 #3
2622 {
2623   \tl_if_eq:nnTF { \stop } { #3 }
2624   {
2625     \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2626     {
2627       \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2628       \tl_gput_right:Nx \g_@@_pre_code_after_tl
2629       { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2630       \tl_gset:Nn \g_@@_right_delim_tl { #2 }
2631     }
2632     {
2633       \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2634       \tl_gput_right:Nx \g_@@_pre_code_after_tl
2635       { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2636       \@@_error:nn { double~closing~delimiter } { #2 }
2637     }
2638   }
2639   {
2640     \tl_gput_right:Nx \g_@@_pre_code_after_tl
2641     { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2642     \@@_error:nn { double~closing~delimiter } { #2 }
2643     \@@_rec_preamble:n #3
2644   }
2645 }
2646 \cs_new:cpn { @@ _ \token_to_str:N \right } #1
2647   { \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 @{...}.

```

2648 \cs_new_protected:Npn \@@_rec_preamble_after_col:n #1
2649 {
2650   \str_if_eq:nnTF { #1 } { < }

```

```

2651 \@@_rec_preamble_after_col_i:n
2652 {
2653   \str_if_eq:nnTF { #1 } { @ }
2654   \@@_rec_preamble_after_col_ii:n
2655   {
2656     \tl_if_eq:NNTF \l_@@_vlines_clist \c_@@_all_tl
2657     {
2658       \tl_gput_right:Nn \g_@@_array_preamble_tl
2659       { ! { \skip_horizontal:N \arrayrulewidth } }
2660     }
2661     {
2662       \exp_args:NNe
2663       \clist_if_in:NnT \l_@@_vlines_clist { \int_eval:n { \c@jCol + 1 } }
2664       {
2665         \tl_gput_right:Nn \g_@@_array_preamble_tl
2666         { ! { \skip_horizontal:N \arrayrulewidth } }
2667       }
2668     }
2669     \@@_rec_preamble:n { #1 }
2670   }
2671 }
2672 }
2673 \cs_new_protected:Npn \@@_rec_preamble_after_col_i:n #1
2674 {
2675   \tl_gput_right:Nn \g_@@_array_preamble_tl { < { #1 } }
2676   \@@_rec_preamble_after_col:n
2677 }

```

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.

```

2678 \cs_new_protected:Npn \@@_rec_preamble_after_col_ii:n #1
2679 {
2680   \tl_if_eq:NNTF \l_@@_vlines_clist \c_@@_all_tl
2681   {
2682     \tl_gput_right:Nn \g_@@_array_preamble_tl
2683     { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2684   }
2685   {
2686     \exp_args:NNe
2687     \clist_if_in:NnTF \l_@@_vlines_clist { \int_eval:n { \c@jCol + 1 } }
2688     {
2689       \tl_gput_right:Nn \g_@@_array_preamble_tl
2690       { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2691     }
2692     { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { #1 } } }
2693   }
2694   \@@_rec_preamble:n
2695 }

```

```

2696 \cs_new:cpn { @@ _ * } #1 #2 #3
2697 {
2698   \tl_clear:N \l_tmpa_tl
2699   \int_step_inline:nn { #2 } { \tl_put_right:Nn \l_tmpa_tl { #3 } }
2700   \exp_last_unbraced:No \@@_rec_preamble:n \l_tmpa_tl
2701 }

```

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.

```

2702 \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.

```

2703 \cs_new:Npn \@@_X #1 #2
2704 {
2705   \str_if_eq:nnTF { #2 } { [ ]
2706     { \@@_make_preamble_X:w [ ]
2707       { \@@_make_preamble_X:w [ ] #2 }
2708     }
2709   \cs_new_protected:Npn \@@_make_preamble_X:w [ #1 ]
2710     { \@@_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 { WithArrows / 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).

```

2711 \keys_define:nn { WithArrows / X-column }
2712 { 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.

```

2713 \cs_new_protected:Npn \@@_make_preamble_X:i:n #1
2714 {

```

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).

```

2715   \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).

```

2716   \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.

```

2717   \int_zero_new:N \l_@@_weight_int
2718   \int_set_eq:NN \l_@@_weight_int \c_one_int
2719   \@@_keys_p_column:n { #1 }

```

The unknown keys are put in \l_tmpa_tl

```

2720   \keys_set:no { WithArrows / X-column } \l_tmpa_tl
2721   \int_compare:nNnT \l_@@_weight_int < \c_zero_int
2722   {
2723     \@@_error_or_warning:n { negative-weight }
2724     \int_set:Nn \l_@@_weight_int { - \l_@@_weight_int }
2725   }
2726   \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).

```

2727   \bool_if:NTF \l_@@_X_columns_aux_bool
2728   {
2729     \exp_args:Nne
2730     \@@_make_preamble_ii_iv:nnn
2731     { \l_@@_weight_int \l_@@_X_columns_dim }
2732     { minipage }
2733     { \@@_no_update_width: }
2734   }
2735   {
2736     \tl_gput_right:Nn \g_@@_array_preamble_tl
2737     {
2738       > {
2739         \@@_cell_begin:w
2740         \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`.

```
2741 \NotEmpty
```

The following code will nullify the box of the cell.

```
2742 \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2743 { \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`.

```
2744 \begin { minipage } { 5 cm } \arraybackslash
2745 }
2746 c
2747 < {
2748 \end { minipage }
2749 \@@_cell_end:
2750 }
2751 }
2752 \int_gincr:N \c@jCol
2753 \@@_rec_preamble_after_col:n
2754 }
2755 }

2756 \cs_new_protected:Npn \@@_no_update_width:
2757 {
2758 \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2759 { \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing: }
2760 }
```

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

```
2761 \cs_new_protected:Npn \@@_make_preamble_vlism:n #1
2762 {
2763 \seq_gput_right:Nx \g_@@_cols_vlism_seq
2764 { \int_eval:n { \c@jCol + 1 } }
2765 \tl_gput_right:Nx \g_@@_array_preamble_tl
2766 { \exp_not:N ! { \skip_horizontal:N \arrayrulewidth } }
2767 \@@_rec_preamble:n
2768 }
```

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.

```
2769 \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).

```
2770 \cs_new_protected:cpn { @@ _ \token_to_str:N \hline }
2771 { \@@_fatal:n { Preamble~forgotten } }
2772 \cs_set_eq:cc { @@ _ \token_to_str:N \Hline } { @@ _ \token_to_str:N \hline }
2773 \cs_set_eq:cc { @@ _ \token_to_str:N \toprule } { @@ _ \token_to_str:N \hline }
2774 \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).

```
2775 \cs_new:Npn \@@_multicolumn:nnn #1 #2 #3
2776 {
```

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 that one column specifier in the preamble of `\multicolumn`.

```

2777 \multispan { #1 }
2778 \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing: % added 2023-10-04
2779 \begingroup
2780 \cs_set:Npn \@addamp
2781 { \legacy_if:nTF { @firstamp } { \@firstampfalse } { \@preamerr 5 } }

```

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

```

2782 \tl_gclear:N \g_@@_preamble_tl
2783 \@@_make_m_preamble:n #2 \q_stop

```

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

```

2784 \exp_args:No \@mkpream \g_@@_preamble_tl
2785 \@addtopreamble \empty
2786 \endgroup

```

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

```

2787 \int_compare:nNnT { #1 } > \c_one_int
2788 {
2789   \seq_gput_left:Nx \g_@@_multicolumn_cells_seq
2790   { \int_use:N \c@iRow - \int_eval:n { \c@jCol + 1 } }
2791   \seq_gput_left:Nn \g_@@_multicolumn_sizes_seq { #1 }
2792   \seq_gput_right:Nx \g_@@_pos_of_blocks_seq
2793   {
2794     {
2795       \int_if_zero:nTF \c@jCol
2796       { \int_eval:n { \c@iRow + 1 } }
2797       { \int_use:N \c@iRow }
2798     }
2799     { \int_eval:n { \c@jCol + 1 } }
2800     {
2801       \int_if_zero:nTF \c@jCol
2802       { \int_eval:n { \c@iRow + 1 } }
2803       { \int_use:N \c@iRow }
2804     }
2805     { \int_eval:n { \c@jCol + #1 } }
2806     { } % for the name of the block
2807   }
2808 }

```

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

```

2809 \cs_set:Npn \@sharp { #3 }
2810 \@arstrut
2811 \@preamble
2812 \null

```

We add some lines.

```

2813 \int_gadd:Nn \c@jCol { #1 - 1 }
2814 \int_compare:nNnT \c@jCol > \g_@@_col_total_int
2815 { \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
2816 \ignorespaces
2817 }

```

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`.

```

2818 \cs_new_protected:Npn \@@_make_m_preamble:n #1
2819 {
2820   \str_case:nnF { #1 }
2821   {

```

```

2822     c { \@@_make_m_preamble_i:n #1 }
2823     l { \@@_make_m_preamble_i:n #1 }
2824     r { \@@_make_m_preamble_i:n #1 }
2825     > { \@@_make_m_preamble_ii:nn #1 }
2826     ! { \@@_make_m_preamble_ii:nn #1 }
2827     @ { \@@_make_m_preamble_ii:nn #1 }
2828     | { \@@_make_m_preamble_iii:n #1 }
2829     p { \@@_make_m_preamble_iv:nnn t #1 }
2830     m { \@@_make_m_preamble_iv:nnn c #1 }
2831     b { \@@_make_m_preamble_iv:nnn b #1 }
2832     w { \@@_make_m_preamble_v:nnnn { } #1 }
2833     W { \@@_make_m_preamble_v:nnnn { \@@_special_W: } #1 }
2834     \q_stop { }
2835   }
2836   {
2837     \cs_if_exist:cTF { NC @ find @ #1 }
2838     {
2839       \tl_set_eq:Nc \l_tmpa_tl { NC @ rewrite @ #1 }
2840       \exp_last_unbraced:No \@@_make_m_preamble:n \l_tmpa_tl
2841     }
2842     {
2843       \tl_if_eq:nnT { #1 } { S }
2844       { \@@_fatal:n { unknown~column~type~S } }
2845       { \@@_fatal:nn { unknown~column~type } { #1 } }
2846     }
2847   }
2848 }

```

For c, l and r

```

2849 \cs_new_protected:Npn \@@_make_m_preamble_i:n #1
2850 {
2851   \tl_gput_right:Nn \g_@@_preamble_tl
2852   {
2853     > { \@@_cell_begin:w \cs_set_nopar:Npn \l_@@_hpos_cell_tl { #1 } }
2854     #1
2855     < \@@_cell_end:
2856   }

```

We test for the presence of a <.

```

2857   \@@_make_m_preamble_x:n
2858 }

```

For >, ! and @

```

2859 \cs_new_protected:Npn \@@_make_m_preamble_ii:nn #1 #2
2860 {
2861   \tl_gput_right:Nn \g_@@_preamble_tl { #1 { #2 } }
2862   \@@_make_m_preamble:n
2863 }

```

For |

```

2864 \cs_new_protected:Npn \@@_make_m_preamble_iii:n #1
2865 {
2866   \tl_gput_right:Nn \g_@@_preamble_tl { #1 }
2867   \@@_make_m_preamble:n
2868 }

```

For p, m and b

```

2869 \cs_new_protected:Npn \@@_make_m_preamble_iv:nnn #1 #2 #3
2870 {
2871   \tl_gput_right:Nn \g_@@_preamble_tl
2872   {
2873     > {
2874       \@@_cell_begin:w

```

```

2875         \begin { minipage } [ #1 ] { \dim_eval:n { #3 } }
2876         \mode_leave_vertical:
2877         \arraybackslash
2878         \vrule height \box_ht:N \@arstrutbox depth 0 pt width 0 pt
2879     }
2880     c
2881     < {
2882         \vrule height 0 pt depth \box_dp:N \@arstrutbox width 0 pt
2883         \end { minipage }
2884         \@@_cell_end:
2885     }
2886 }

```

We test for the presence of a <.

```

2887     \@@_make_m_preamble_x:n
2888 }

```

For w and W

```

2889 \cs_new_protected:Npn \@@_make_m_preamble_v:nnnn #1 #2 #3 #4
2890 {
2891     \tl_gput_right:Nn \g_@@_preamble_tl
2892     {
2893         > {
2894             \dim_set:Nn \l_@@_col_width_dim { #4 }
2895             \hbox_set:Nw \l_@@_cell_box
2896             \@@_cell_begin:w
2897             \cs_set_nopar:Npn \l_@@_hpos_cell_tl { #3 }
2898         }
2899         c
2900         < {
2901             \@@_cell_end:
2902             \hbox_set_end:
2903             \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
2904             #1
2905             \@@_adjust_size_box:
2906             \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2907         }
2908     }

```

We test for the presence of a <.

```

2909     \@@_make_m_preamble_x:n
2910 }

```

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

```

2911 \cs_new_protected:Npn \@@_make_m_preamble_x:n #1
2912 {
2913     \str_if_eq:nnTF { #1 } { < }
2914     \@@_make_m_preamble_ix:n
2915     { \@@_make_m_preamble:n { #1 } }
2916 }
2917 \cs_new_protected:Npn \@@_make_m_preamble_ix:n #1
2918 {
2919     \tl_gput_right:Nn \g_@@_preamble_tl { < { #1 } }
2920     \@@_make_m_preamble_x:n
2921 }

```

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).

```

2922 \cs_new_protected:Npn \@@_put_box_in_flow:
2923 {
2924     \box_set_ht:Nn \l_tmpa_box { \box_ht:N \l_tmpa_box + \l_tmpa_dim }

```

```

2925 \box_set_dp:Nn \l_tmpa_box { \box_dp:N \l_tmpa_box + \l_tmpb_dim }
2926 \tl_if_eq:NNTF \l_@@_baseline_tl \c_@@_c_tl
2927 { \box_use_drop:N \l_tmpa_box }
2928 \@@_put_box_in_flow_i:
2929 }

```

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).

```

2930 \cs_new_protected:Npn \@@_put_box_in_flow_i:
2931 {
2932 \pgfpicture
2933 \@@_qpoint:n { row - 1 }
2934 \dim_gset_eq:NN \g_tmpa_dim \pgf@y
2935 \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
2936 \dim_gadd:Nn \g_tmpa_dim \pgf@y
2937 \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).

```

2938 \tl_if_in:NnTF \l_@@_baseline_tl { line- }
2939 {
2940 \int_set:Nn \l_tmpa_int
2941 {
2942 \str_range:Nnn
2943 \l_@@_baseline_tl
2944 6
2945 { \tl_count:o \l_@@_baseline_tl }
2946 }
2947 \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
2948 }
2949 {
2950 \tl_if_eq:NnTF \l_@@_baseline_tl { t }
2951 { \int_set_eq:NN \l_tmpa_int \c_one_int }
2952 {
2953 \tl_if_eq:NnTF \l_@@_baseline_tl { b }
2954 { \int_set_eq:NN \l_tmpa_int \c@iRow }
2955 { \int_set:Nn \l_tmpa_int \l_@@_baseline_tl }
2956 }
2957 \bool_lazy_or:nnT
2958 { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
2959 { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
2960 {
2961 \@@_error:n { bad-value-for-baseline }
2962 \int_set_eq:NN \l_tmpa_int \c_one_int
2963 }
2964 \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }

```

We take into account the position of the mathematical axis.

```

2965 \dim_gsub:Nn \g_tmpa_dim { \fontdimen22 \textfont2 }
2966 }
2967 \dim_gsub:Nn \g_tmpa_dim \pgf@y

```

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

```

2968 \endpgfpicture
2969 \box_move_up:nn \g_tmpa_dim { \box_use_drop:N \l_tmpa_box }
2970 \box_use_drop:N \l_tmpa_box
2971 }

```

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).

```

2972 \cs_new_protected:Npn \@@_use_arraybox_with_notes_c:
2973 {

```

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.

```

2974 \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
2975 {
2976   \int_compare:nNnT \c@jCol > \c_one_int
2977   {
2978     \box_set_wd:Nn \l_@@_the_array_box
2979     { \box_wd:N \l_@@_the_array_box - \arraycolsep }
2980   }
2981 }

```

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

```

2982 \begin { minipage } [ t ] { \box_wd:N \l_@@_the_array_box }
2983 \bool_if:NT \l_@@_caption_above_bool
2984 {
2985   \tl_if_empty:NF \l_@@_caption_tl
2986   {
2987     \bool_set_false:N \g_@@_caption_finished_bool
2988     \int_gzero:N \c@tabularnote
2989     \@@_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).

```

2990 \int_compare:nNnT \g_@@_notes_caption_int > \c_zero_int
2991 {
2992   \tl_gput_right:Nx \g_@@_aux_tl
2993   {
2994     \tl_set:Nn \exp_not:N \l_@@_note_in_caption_tl
2995     { \int_use:N \g_@@_notes_caption_int }
2996   }
2997   \int_gzero:N \g_@@_notes_caption_int
2998 }
2999 }
3000 }

```

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

```

3001 \hbox
3002 {
3003   \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.

```

3004 \@@_create_extra_nodes:
3005 \seq_if_empty:NF \g_@@_blocks_seq \@@_draw_blocks:
3006 }

```

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).

```

3007 \bool_lazy_any:nT
3008 {
3009   { ! \seq_if_empty_p:N \g_@@_notes_seq }
3010   { ! \seq_if_empty_p:N \g_@@_notes_in_caption_seq }
3011   { ! \tl_if_empty_p:o \g_@@_tabularnote_tl }
3012 }
3013 \@@_insert_tabularnotes:
3014 \cs_set_eq:NN \tabularnote \@@_tabularnote_error:n
3015 \bool_if:NF \l_@@_caption_above_bool \@@_insert_caption:

```

```

3016 \end { minipage }
3017 }

```

```

3018 \cs_new_protected:Npn \@@_insert_caption:
3019 {
3020 \tl_if_empty:NF \l_@@_caption_tl
3021 {
3022 \cs_if_exist:NTF \@capttype
3023 { \@@_insert_caption_i: }
3024 { \@@_error:n { caption~outside~float } }
3025 }
3026 }

```

```

3027 \cs_new_protected:Npn \@@_insert_caption_i:
3028 {
3029 \group_begin:

```

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

```

3030 \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.

```

3031 \IfPackageLoadedTF { floatrow }
3032 { \cs_set_eq:NN \@makecaption \FR@makecaption }
3033 { }
3034 \tl_if_empty:NTF \l_@@_short_caption_tl
3035 { \caption }
3036 { \caption [ \l_@@_short_caption_tl ] }
3037 { \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.

```

3038 \bool_if:NF \g_@@_caption_finished_bool
3039 {
3040 \bool_gset_true:N \g_@@_caption_finished_bool
3041 \int_gset_eq:NN \g_@@_notes_caption_int \c@tabularnote
3042 \int_gzero:N \c@tabularnote
3043 }
3044 \tl_if_empty:NF \l_@@_label_tl { \label { \l_@@_label_tl } }
3045 \group_end:
3046 }

```

```

3047 \cs_new_protected:Npn \@@_tabularnote_error:n #1
3048 {
3049 \@@_error_or_warning:n { tabularnote~below~the~tabular }
3050 \@@_gredirect_none:n { tabularnote~below~the~tabular }
3051 }
3052 \cs_new_protected:Npn \@@_insert_tabularnotes:
3053 {
3054 \seq_gconcat:NNN \g_@@_notes_seq \g_@@_notes_in_caption_seq \g_@@_notes_seq
3055 \int_set:Nn \c@tabularnote { \seq_count:N \g_@@_notes_seq }
3056 \skip_vertical:N 0.65ex

```

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

```

3057 \group_begin:
3058 \l_@@_notes_code_before_tl
3059 \tl_if_empty:NF \g_@@_tabularnote_tl
3060 {

```

```

3061     \g_@@_tabularnote_tl \par
3062     \tl_gclear:N \g_@@_tabularnote_tl
3063 }

```

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.

```

3064     \int_compare:nNnT \c@tabularnote > \c_zero_int
3065     {
3066         \bool_if:NTF \l_@@_notes_para_bool
3067         {
3068             \begin { tabularnotes* }
3069             \seq_map_inline:Nn \g_@@_notes_seq
3070             { \@@_one_tabularnote:nn ##1 }
3071             \strut
3072             \end { tabularnotes* }

```

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

```

3073         \par
3074     }
3075     {
3076         \tabularnotes
3077         \seq_map_inline:Nn \g_@@_notes_seq
3078         { \@@_one_tabularnote:nn ##1 }
3079         \strut
3080         \endtabularnotes
3081     }
3082 }
3083 \unskip
3084 \group_end:
3085 \bool_if:NT \l_@@_notes_bottomrule_bool
3086 {
3087     \IfPackageLoadedTF { booktabs }
3088     {

```

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

```

3089         \skip_vertical:N \aboverulesep

```

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

```

3090         { \CT@arc@ \hrule height \heavyrulewidth }
3091     }
3092     { \@@_error_or_warning:n { bottomrule~without~booktabs } }
3093 }
3094 \l_@@_notes_code_after_tl
3095 \seq_gclear:N \g_@@_notes_seq
3096 \seq_gclear:N \g_@@_notes_in_caption_seq
3097 \int_gzero:N \c@tabularnote
3098 }

```

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 curryfication.

```

3099 \cs_set_protected:Npn \@@_one_tabularnote:nn #1
3100 {
3101     \tl_if_novalue:nTF { #1 }
3102     { \item }
3103     { \item [ \@@_notes_label_in_list:n { #1 } ] }
3104 }

```

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`.

```

3105 \cs_new_protected:Npn \@@_use_arraybox_with_notes_b:

```

```

3106 {
3107   \pgfpicture
3108     \@@_qpoint:n { row - 1 }
3109     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3110     \@@_qpoint:n { row - \int_use:N \c@iRow - base }
3111     \dim_gsub:Nn \g_tmpa_dim \pgf@y
3112   \endpgfpicture
3113   \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3114   \int_if_zero:nT \l_@@_first_row_int
3115   {
3116     \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3117     \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3118   }
3119   \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3120 }

```

Now, the general case.

```

3121 \cs_new_protected:Npn \@@_use_arraybox_with_notes:
3122 {

```

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

```

3123   \tl_if_eq:NnT \l_@@_baseline_tl { t }
3124   { \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`.

```

3125   \pgfpicture
3126   \@@_qpoint:n { row - 1 }
3127   \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3128   \str_if_in:NnTF \l_@@_baseline_tl { line- }
3129   {
3130     \int_set:Nn \l_tmpa_int
3131     {
3132       \str_range:Nnn
3133         \l_@@_baseline_tl
3134         6
3135       { \tl_count:o \l_@@_baseline_tl }
3136     }
3137     \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
3138   }
3139   {
3140     \int_set:Nn \l_tmpa_int \l_@@_baseline_tl
3141     \bool_lazy_or:nnT
3142       { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
3143       { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
3144     {
3145       \@@_error:n { bad-value-for-baseline }
3146       \int_set:Nn \l_tmpa_int 1
3147     }
3148     \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }
3149   }
3150   \dim_gsub:Nn \g_tmpa_dim \pgf@y
3151   \endpgfpicture
3152   \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3153   \int_if_zero:nT \l_@@_first_row_int
3154   {
3155     \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3156     \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3157   }
3158   \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3159 }

```

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.

```

3160 \cs_new_protected:Npn \@@_put_box_in_flow_bis:nn #1 #2
3161 {

```

We will compute the real width of both delimiters used.

```

3162 \dim_zero_new:N \l_@@_real_left_delim_dim
3163 \dim_zero_new:N \l_@@_real_right_delim_dim
3164 \hbox_set:Nn \l_tmpb_box
3165 {
3166   \c_math_toggle_token
3167   \left #1
3168   \vcenter
3169   {
3170     \vbox_to_ht:nn
3171     { \box_ht_plus_dp:N \l_tmpa_box }
3172     { }
3173   }
3174   \right .
3175   \c_math_toggle_token
3176 }
3177 \dim_set:Nn \l_@@_real_left_delim_dim
3178 { \box_wd:N \l_tmpb_box - \nulldelimiterspace }
3179 \hbox_set:Nn \l_tmpb_box
3180 {
3181   \c_math_toggle_token
3182   \left .
3183   \vbox_to_ht:nn
3184   { \box_ht_plus_dp:N \l_tmpa_box }
3185   { }
3186   \right #2
3187   \c_math_toggle_token
3188 }
3189 \dim_set:Nn \l_@@_real_right_delim_dim
3190 { \box_wd:N \l_tmpb_box - \nulldelimiterspace }

```

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

```

3191 \skip_horizontal:N \l_@@_left_delim_dim
3192 \skip_horizontal:N -\l_@@_real_left_delim_dim
3193 \@@_put_box_in_flow:
3194 \skip_horizontal:N \l_@@_right_delim_dim
3195 \skip_horizontal:N -\l_@@_real_right_delim_dim
3196 }

```

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).

```

3197 \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).

```

3198 {
3199   \peek_remove_spaces:n
3200   {
3201     \peek_meaning:NTF \end
3202     \@@_analyze_end:Nn
3203     {
3204       \@@_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`).

```

3205         \exp_args:No \@@_array: \g_@@_array_preamble_tl
3206     }
3207 }
3208 }
3209 {
3210     \@@_create_col_nodes:
3211     \endarray
3212 }
```

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

```

3213 \NewDocumentEnvironment { @@-light-syntax } { b }
3214 {
```

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`.

```

3215     \tl_if_empty:nT { #1 } { \@@_fatal:n { empty-environment } }
3216     \tl_map_inline:nn { #1 }
3217     {
3218         \str_if_eq:nnT { ##1 } { & }
3219         { \@@_fatal:n { ampersand-in-light-syntax } }
3220         \str_if_eq:nnT { ##1 } { \ }
3221         { \@@_fatal:n { double-backslash-in-light-syntax } }
3222     }
```

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`.

```

3223     \@@_light_syntax_i:w #1 \CodeAfter \q_stop
```

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

```

3224 }
```

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.

```

3225 {
3226     \@@_create_col_nodes:
3227     \endarray
3228 }
3229 \cs_new_protected:Npn \@@_light_syntax_i:w #1\CodeAfter #2\q_stop
3230 {
3231     \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).

```

3232     \seq_clear_new:N \l_@@_rows_seq
```

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

```

3233     \tl_set_rescan:Nno \l_@@_end_of_row_tl { } \l_@@_end_of_row_tl
3234     \bool_if:NTF \l_@@_light_syntax_expanded_bool
3235         \seq_set_split:Nee
3236         \seq_set_split:NVn
3237         \l_@@_rows_seq \l_@@_end_of_row_tl { #1 }
```

We delete the last row if it is empty.

```

3238     \seq_pop_right:NN \l_@@_rows_seq \l_tmpa_tl
3239     \tl_if_empty:NF \l_tmpa_tl
3240     { \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.

```

3241 \int_compare:nNnT \l_@@_last_row_int = { -1 }
3242 { \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`).

```

3243 \tl_build_begin:N \l_@@_new_body_tl
3244 \int_zero_new:N \l_@@_nb_cols_int

```

First, we treat the first row.

```

3245 \seq_pop_left:NN \l_@@_rows_seq \l_tmpa_tl
3246 \@@_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).

```

3247 \seq_map_inline:Nn \l_@@_rows_seq
3248 {
3249   \tl_build_put_right:Nn \l_@@_new_body_tl { \\ }
3250   \@@_line_with_light_syntax:n { #1 }
3251 }
3252 \tl_build_end:N \l_@@_new_body_tl
3253 \int_compare:nNnT \l_@@_last_col_int = { -1 }
3254 {
3255   \int_set:Nn \l_@@_last_col_int
3256   { \l_@@_nb_cols_int - 1 + \l_@@_first_col_int }
3257 }

```

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.

```

3258 \@@_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`).

```

3259 \exp_args:No \@@_array: \g_@@_array_preamble_tl \l_@@_new_body_tl
3260 }
3261 \cs_new_protected:Npn \@@_line_with_light_syntax:n #1
3262 {
3263   \seq_clear_new:N \l_@@_cells_seq
3264   \seq_set_split:Nnn \l_@@_cells_seq { ~ } { #1 }
3265   \int_set:Nn \l_@@_nb_cols_int
3266   {
3267     \int_max:nn
3268     \l_@@_nb_cols_int
3269     { \seq_count:N \l_@@_cells_seq }
3270   }
3271   \seq_pop_left:NN \l_@@_cells_seq \l_tmpa_tl
3272   \exp_args:NNo \tl_build_put_right:Nn \l_@@_new_body_tl \l_tmpa_tl
3273   \seq_map_inline:Nn \l_@@_cells_seq
3274   { \tl_build_put_right:Nn \l_@@_new_body_tl { & #1 } }
3275 }
3276 \cs_generate_variant:Nn \@@_line_with_light_syntax:n { o }

```

The following command is used by the code which detects whether the environment is empty (we raise a fatal error in this case: it's only a security). When this command is used, `#1` is, in fact, always `\end`.

```

3277 \cs_new_protected:Npn \@@_analyze_end:Nn #1 #2
3278 {
3279   \str_if_eq:onT \g_@@_name_env_str { #2 }
3280   { \@@_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.

```
3281   \end { #2 }
3282 }
```

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).

```
3283 \cs_new:Npn \@@_create_col_nodes:
3284 {
3285   \crr
3286   \int_if_zero:nT \l_@@_first_col_int
3287   {
3288     \omit
3289     \hbox_overlap_left:n
3290     {
3291       \bool_if:NT \l_@@_code_before_bool
3292       { \pgfsys@markposition { \@@_env: - col - 0 } }
3293       \pgfpicture
3294       \pgfrememberpicturepositiononpagetrue
3295       \pgfcoordinate { \@@_env: - col - 0 } \pgfpintorigin
3296       \str_if_empty:NF \l_@@_name_str
3297       { \pgfnodealias { \l_@@_name_str - col - 0 } { \@@_env: - col - 0 } }
3298       \endpgfpicture
3299       \skip_horizontal:N 2\col@sep
3300       \skip_horizontal:N \g_@@_width_first_col_dim
3301     }
3302     &
3303   }
3304   \omit
```

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

```
3305   \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`).

```
3306   \int_if_zero:nTF \l_@@_first_col_int
3307   {
3308     \bool_if:NT \l_@@_code_before_bool
3309     {
3310       \hbox
3311       {
3312         \skip_horizontal:N -0.5\arrayrulewidth
3313         \pgfsys@markposition { \@@_env: - col - 1 }
3314         \skip_horizontal:N 0.5\arrayrulewidth
3315       }
3316     }
3317     \pgfpicture
3318     \pgfrememberpicturepositiononpagetrue
3319     \pgfcoordinate { \@@_env: - col - 1 }
3320     { \pgfpint { - 0.5 \arrayrulewidth } \c_zero_dim }
3321     \str_if_empty:NF \l_@@_name_str
3322     { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
3323     \endpgfpicture
3324   }
3325   {
3326     \bool_if:NT \l_@@_code_before_bool
3327     {
3328       \hbox
3329       {
3330         \skip_horizontal:N 0.5\arrayrulewidth
3331         \pgfsys@markposition { \@@_env: - col - 1 }
3332         \skip_horizontal:N -0.5\arrayrulewidth
```

```

3333     }
3334   }
3335   \pgfpicture
3336   \pgfrememberpicturepositiononpagetrue
3337   \pgfcoordinate { \@@_env: - col - 1 }
3338   { \pgfpoint { 0.5 \arrayrulewidth } \c_zero_dim }
3339   \str_if_empty:NF \l_@@_name_str
3340   { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
3341   \endpgfpicture
3342 }

```

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.

```

3343   \skip_gset:Nn \g_tmpa_skip { 0 pt~plus 1 fill }
3344   \bool_if:NF \l_@@_auto_columns_width_bool
3345   { \dim_compare:nNnT \l_@@_columns_width_dim > \c_zero_dim }
3346   {
3347     \bool_lazy_and:nnTF
3348     \l_@@_auto_columns_width_bool
3349     { \bool_not_p:n \l_@@_block_auto_columns_width_bool }
3350     { \skip_gadd:Nn \g_tmpa_skip \g_@@_max_cell_width_dim }
3351     { \skip_gadd:Nn \g_tmpa_skip \l_@@_columns_width_dim }
3352     \skip_gadd:Nn \g_tmpa_skip { 2 \col@sep }
3353   }
3354   \skip_horizontal:N \g_tmpa_skip
3355   \hbox
3356   {
3357     \bool_if:NT \l_@@_code_before_bool
3358     {
3359       \hbox
3360       {
3361         \skip_horizontal:N -0.5\arrayrulewidth
3362         \pgfsys@markposition { \@@_env: - col - 2 }
3363         \skip_horizontal:N 0.5\arrayrulewidth
3364       }
3365     }
3366     \pgfpicture
3367     \pgfrememberpicturepositiononpagetrue
3368     \pgfcoordinate { \@@_env: - col - 2 }
3369     { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3370     \str_if_empty:NF \l_@@_name_str
3371     { \pgfnodealias { \l_@@_name_str - col - 2 } { \@@_env: - col - 2 } }
3372     \endpgfpicture
3373   }

```

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.

```

3374   \int_gset_eq:NN \g_tmpa_int \c_one_int
3375   \bool_if:NTF \g_@@_last_col_found_bool
3376   { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 3 } \c_zero_int } }
3377   { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 2 } \c_zero_int } }
3378   {
3379     &
3380     \omit
3381     \int_gincr:N \g_tmpa_int

```

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

```

3382     \skip_horizontal:N \g_tmpa_skip
3383     \bool_if:NT \l_@@_code_before_bool
3384     {

```

```

3385         \hbox
3386         {
3387             \skip_horizontal:N -0.5\arrayrulewidth
3388             \pgfsys@markposition
3389             { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3390             \skip_horizontal:N 0.5\arrayrulewidth
3391         }
3392     }

```

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

```

3393     \pgfpicture
3394     \pgfrememberpicturepositiononpagetrue
3395     \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3396     { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3397     \str_if_empty:NF \l_@@_name_str
3398     {
3399         \pgfnodealias
3400         { \l_@@_name_str - col - \int_eval:n { \g_tmpa_int + 1 } }
3401         { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3402     }
3403     \endpgfpicture
3404 }

```

```

3405     &
3406     \omit

```

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

```

3407     \int_if_zero:nT \g_@@_col_total_int
3408     { \skip_gset:Nn \g_tmpa_skip { 0 pt+plus 1 fill } }
3409     \skip_horizontal:N \g_tmpa_skip
3410     \int_gincr:N \g_tmpa_int
3411     \bool_lazy_any:nF % modified 2023/12/13
3412     {
3413         \g_@@_delims_bool
3414         \l_@@_tabular_bool
3415         { ! \clist_if_empty_p:N \l_@@_vlines_clist }
3416         \l_@@_exterior_arraycolsep_bool
3417         \l_@@_bar_at_end_of_pream_bool
3418     }
3419     { \skip_horizontal:N -\col@sep }
3420     \bool_if:NT \l_@@_code_before_bool
3421     {
3422         \hbox
3423         {
3424             \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.

```

3425         \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3426         { \skip_horizontal:N -\arraycolsep }
3427         \pgfsys@markposition
3428         { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3429         \skip_horizontal:N 0.5\arrayrulewidth
3430         \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3431         { \skip_horizontal:N \arraycolsep }
3432     }
3433 }
3434 \pgfpicture
3435 \pgfrememberpicturepositiononpagetrue
3436 \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3437 {
3438     \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool

```

```

3439         {
3440             \pgfpoint
3441             { - 0.5 \arrayrulewidth - \arraycolsep }
3442             \c_zero_dim
3443         }
3444         { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3445     }
3446     \str_if_empty:NF \l_@@_name_str
3447     {
3448         \pgfnodealias
3449         { \l_@@_name_str - col - \int_eval:n { \g_tmpa_int + 1 } }
3450         { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3451     }
3452     \endpgfpicture

3453     \bool_if:NT \g_@@_last_col_found_bool
3454     {
3455         \hbox_overlap_right:n
3456         {
3457             \skip_horizontal:N \g_@@_width_last_col_dim
3458             \skip_horizontal:N \col@sep % added 2023-11-05
3459             \bool_if:NT \l_@@_code_before_bool
3460             {
3461                 \pgfsys@markposition
3462                 { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3463             }
3464             \pgfpicture
3465             \pgfrememberpicturepositiononpagetrue
3466             \pgfcoordinate
3467             { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3468             \pgfpointorigin
3469             \str_if_empty:NF \l_@@_name_str
3470             {
3471                 \pgfnodealias
3472                 {
3473                     \l_@@_name_str - col
3474                     - \int_eval:n { \g_@@_col_total_int + 1 }
3475                 }
3476                 { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3477             }
3478             \endpgfpicture
3479         }
3480     }
3481     \cr
3482 }

```

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

```

3483 \tl_const:Nn \c_@@_preamble_first_col_tl
3484 {
3485     >
3486     {

```

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).

```

3487     \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
3488     \bool_gset_true:N \g_@@_after_col_zero_bool
3489     \@@_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.

```

3490     \hbox_set:Nw \l_@@_cell_box
3491     \@@_math_toggle:
3492     \@@_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”.

```

3493     \int_compare:nNnT \c@iRow > \c_zero_int
3494     {
3495         \bool_lazy_or:nnT
3496         { \int_compare_p:nNn \l_@@_last_row_int < \c_zero_int }
3497         { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
3498         {
3499             \l_@@_code_for_first_col_tl
3500             \xglobal \colorlet { nicematrix-first-col } { . }
3501         }
3502     }
3503 }
```

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`.

```

3504     l
3505     <
3506     {
3507         \@@_math_toggle:
3508         \hbox_set_end:
3509         \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
3510         \@@_adjust_size_box:
3511         \@@_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.

```

3512     \dim_gset:Nn \g_@@_width_first_col_dim
3513     { \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.

```

3514     \hbox_overlap_left:n
3515     {
3516         \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > \c_zero_dim
3517         \@@_node_for_cell:
3518         { \box_use_drop:N \l_@@_cell_box }
3519         \skip_horizontal:N \l_@@_left_delim_dim
3520         \skip_horizontal:N \l_@@_left_margin_dim
3521         \skip_horizontal:N \l_@@_extra_left_margin_dim
3522     }
3523     \bool_gset_false:N \g_@@_empty_cell_bool
3524     \skip_horizontal:N -2\col@sep
3525 }
3526 }
```

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

```

3527 \tl_const:Nn \c_@@_preamble_last_col_tl
3528 {
3529     >
3530     {
3531         \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).

```

3532     \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.

```

3533     \bool_gset_true:N \g_@@_last_col_found_bool
3534     \int_gincr:N \c@jCol
3535     \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.

```

3536     \hbox_set:Nw \l_@@_cell_box
3537     \@@_math_toggle:
3538     \@@_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”.

```

3539     \int_compare:nNnT \c@iRow > \c_zero_int
3540     {
3541         \bool_lazy_or:nnT
3542         { \int_compare_p:nNn \l_@@_last_row_int < \c_zero_int }
3543         { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
3544         {
3545             \l_@@_code_for_last_col_tl
3546             \xglobal \colorlet { nicematrix-last-col } { . }
3547         }
3548     }
3549 }
3550 1
3551 <
3552 {
3553     \@@_math_toggle:
3554     \hbox_set_end:
3555     \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
3556     \@@_adjust_size_box:
3557     \@@_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.

```

3558     \dim_gset:Nn \g_@@_width_last_col_dim
3559     { \dim_max:nn \g_@@_width_last_col_dim { \box_wd:N \l_@@_cell_box } }
3560     \skip_horizontal:N -2\col@sep

```

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

```

3561     \hbox_overlap_right:n
3562     {
3563         \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \c_zero_dim
3564         {
3565             \skip_horizontal:N \l_@@_right_delim_dim
3566             \skip_horizontal:N \l_@@_right_margin_dim
3567             \skip_horizontal:N \l_@@_extra_right_margin_dim
3568             \@@_node_for_cell:
3569         }
3570     }
3571     \bool_gset_false:N \g_@@_empty_cell_bool
3572 }
3573 }

```

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

```

3574 \NewDocumentEnvironment { NiceArray } { }
3575 {
3576     \bool_gset_false:N \g_@@_delims_bool
3577     \str_if_empty:NT \g_@@_name_env_str
3578     { \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).

```

3579     \NiceArrayWithDelims . .
3580 }
3581 { \endNiceArrayWithDelims }

```

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

```

3582 \cs_new_protected:Npn \@@_def_env:nnn #1 #2 #3
3583 {
3584     \NewDocumentEnvironment { #1 NiceArray } { }
3585     {

```

```

3586 \bool_gset_true:N \g_@@_delims_bool
3587 \str_if_empty:NT \g_@@_name_env_str
3588 { \str_gset:Nn \g_@@_name_env_str { #1 NiceArray } }
3589 \@@_test_if_math_mode:
3590 \NiceArrayWithDelims #2 #3
3591 }
3592 { \endNiceArrayWithDelims }
3593 }
3594 \@@_def_env:nnn p ( )
3595 \@@_def_env:nnn b [ ]
3596 \@@_def_env:nnn B \{ \}
3597 \@@_def_env:nnn v | |
3598 \@@_def_env:nnn V \l \l

```

14 The environment `{NiceMatrix}` and its variants

```

3599 \cs_new_protected:Npn \@@_begin_of_NiceMatrix:nn #1 #2
3600 {
3601   \bool_set_false:N \l_@@_preamble_bool
3602   \tl_clear:N \l_tmpa_tl
3603   \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3604   { \tl_set:Nn \l_tmpa_tl { @ { } } }
3605   \tl_put_right:Nn \l_tmpa_tl
3606   {
3607     *
3608     {
3609       \int_case:nnF \l_@@_last_col_int
3610       {
3611         { -2 } { \c@MaxMatrixCols }
3612         { -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).

```

3613       }
3614       { \int_eval:n { \l_@@_last_col_int - 1 } }
3615     }
3616     { #2 }
3617   }
3618   \tl_set:Nn \l_tmpb_tl { \use:c { #1 NiceArray } }
3619   \exp_args:No \l_tmpb_tl \l_tmpa_tl
3620 }
3621 \cs_generate_variant:Nn \@@_begin_of_NiceMatrix:nn { n V }
3622 \clist_map_inline:nn { p , b , B , v , V }
3623 {
3624   \NewDocumentEnvironment { #1 NiceMatrix } { ! O { } }
3625   {
3626     \bool_gset_true:N \g_@@_delims_bool
3627     \str_gset:Nn \g_@@_name_env_str { #1 NiceMatrix }
3628     % added 2023/10/01
3629     \int_if_zero:nT \l_@@_last_col_int
3630     {
3631       \bool_set_true:N \l_@@_last_col_without_value_bool
3632       \int_set:Nn \l_@@_last_col_int { -1 }
3633     }
3634     \keys_set:nn { NiceMatrix / NiceMatrix } { ##1 }
3635     \@@_begin_of_NiceMatrix:nV { #1 } \l_@@_columns_type_tl
3636   }
3637   { \use:c { end #1 NiceArray } }
3638 }

```

We define also an environment `{NiceMatrix}`

```

3639 \NewDocumentEnvironment { NiceMatrix } { ! O { } }
3640 {
3641   \str_gset:Nn \g_@@_name_env_str { NiceMatrix }
3642   % added 2023/10/01
3643   \int_if_zero:nT \l_@@_last_col_int
3644   {
3645     \bool_set_true:N \l_@@_last_col_without_value_bool
3646     \int_set:Nn \l_@@_last_col_int { -1 }
3647   }
3648   \keys_set:nn { NiceMatrix / NiceMatrix } { #1 }
3649   \bool_lazy_or:nnT
3650     { \clist_if_empty_p:N \l_@@_vlines_clist }
3651     { \l_@@_except_borders_bool }
3652     { \bool_set_true:N \l_@@_NiceMatrix_without_vlines_bool }
3653   \@@_begin_of_NiceMatrix:nV { } \l_@@_columns_type_tl
3654 }
3655 { \endNiceArray }

```

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

```

3656 \cs_new_protected:Npn \@@_NotEmpty:
3657 { \bool_gset_true:N \g_@@_not_empty_cell_bool }

```

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

```

3658 \NewDocumentEnvironment { NiceTabular } { 0 { } m ! O { } }
3659 {

```

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`.

```

3660   \dim_compare:nNnT \l_@@_width_dim = \c_zero_dim
3661     { \dim_set_eq:NN \l_@@_width_dim \linewidth }
3662   \str_gset:Nn \g_@@_name_env_str { NiceTabular }
3663   \keys_set:nn { NiceMatrix / NiceTabular } { #1 , #3 }
3664   \tl_if_empty:NF \l_@@_short_caption_tl
3665   {
3666     \tl_if_empty:NT \l_@@_caption_tl
3667     {
3668       \@@_error_or_warning:n { short-caption~without~caption }
3669       \tl_set_eq:NN \l_@@_caption_tl \l_@@_short_caption_tl
3670     }
3671   }
3672   \tl_if_empty:NF \l_@@_label_tl
3673   {
3674     \tl_if_empty:NT \l_@@_caption_tl
3675     { \@@_error_or_warning:n { label~without~caption } }
3676   }
3677   \NewDocumentEnvironment { TabularNote } { b }
3678   {
3679     \bool_if:NTF \l_@@_in_code_after_bool
3680       { \@@_error_or_warning:n { TabularNote~in~CodeAfter } }
3681       {
3682         \tl_if_empty:NF \g_@@_tabularnote_tl
3683         { \tl_gput_right:Nn \g_@@_tabularnote_tl { \par } }
3684         \tl_gput_right:Nn \g_@@_tabularnote_tl { ##1 }
3685       }
3686     }
3687   { }
3688   \@@_settings_for_tabular:
3689   \NiceArray { #2 }
3690 }
3691 { \endNiceArray }

```

```

3692 \cs_new_protected:Npn \@@_settings_for_tabular:
3693 {
3694   \bool_set_true:N \l_@@_tabular_bool
3695   \cs_set_eq:NN \@@_math_toggle: \prg_do_nothing:
3696   \cs_set_eq:NN \@@_tuning_not_tabular_begin: \prg_do_nothing:
3697   \cs_set_eq:NN \@@_tuning_not_tabular_end: \prg_do_nothing:
3698 }

3699 \NewDocumentEnvironment { NiceTabularX } { m O { } m ! O { } }
3700 {
3701   \str_gset:Nn \g_@@_name_env_str { NiceTabularX }
3702   \dim_zero_new:N \l_@@_width_dim
3703   \dim_set:Nn \l_@@_width_dim { #1 }
3704   \keys_set:nn { NiceMatrix / NiceTabular } { #2 , #4 }
3705   \@@_settings_for_tabular:
3706   \NiceArray { #3 }
3707 }
3708 {
3709   \endNiceArray
3710   \int_if_zero:nT \g_@@_total_X_weight_int
3711     { \@@_error:n { NiceTabularX~without~X } }
3712 }

3713 \NewDocumentEnvironment { NiceTabular* } { m O { } m ! O { } }
3714 {
3715   \str_gset:Nn \g_@@_name_env_str { NiceTabular* }
3716   \dim_set:Nn \l_@@_tabular_width_dim { #1 }
3717   \keys_set:nn { NiceMatrix / NiceTabular } { #2 , #4 }
3718   \@@_settings_for_tabular:
3719   \NiceArray { #3 }
3720 }
3721 { \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`).

```

3722 \cs_new_protected:Npn \@@_deal_with_rounded_corners:
3723 {
3724   \bool_lazy_all:nT
3725   {
3726     { \int_compare_p:nNn \l_@@_tab_rounded_corners_dim > \c_zero_dim }
3727     \l_@@_hvlines_bool
3728     { ! \g_@@_delims_bool }
3729     { ! \l_@@_except_borders_bool }
3730   }
3731   {
3732     \bool_set_true:N \l_@@_except_borders_bool
3733     \clist_if_empty:NF \l_@@_corners_clist
3734       { \@@_error:n { hvlines,~rounded-corners~and~corners } }
3735     \tl_gput_right:Nn \g_@@_pre_code_after_tl
3736       {
3737         \@@_stroke_block:nnn
3738         {
3739           rounded-corners = \dim_use:N \l_@@_tab_rounded_corners_dim ,
3740           draw = \l_@@_rules_color_tl
3741         }
3742         { 1-1 }

```

```

3743         { \int_use:N \c@iRow - \int_use:N \c@jCol }
3744     }
3745 }
3746 }

```

```

3747 \cs_new_protected:Npn \@@_after_array:
3748 {
3749     \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.

```

3750     \bool_if:NT \g_@@_last_col_found_bool
3751     { \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`.

```

3752     \bool_if:NT \l_@@_last_col_without_value_bool
3753     { \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.

```

3754     \bool_if:NT \l_@@_last_row_without_value_bool
3755     { \int_set_eq:NN \l_@@_last_row_int \g_@@_row_total_int }

```

```

3756 \tl_gput_right:Nx \g_@@_aux_tl
3757 {
3758     \seq_gset_from_clist:Nn \exp_not:N \g_@@_size_seq
3759     {
3760         \int_use:N \l_@@_first_row_int ,
3761         \int_use:N \c@iRow ,
3762         \int_use:N \g_@@_row_total_int ,
3763         \int_use:N \l_@@_first_col_int ,
3764         \int_use:N \c@jCol ,
3765         \int_use:N \g_@@_col_total_int
3766     }
3767 }

```

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`).

```

3768     \seq_if_empty:NF \g_@@_pos_of_blocks_seq
3769     {
3770         \tl_gput_right:Nx \g_@@_aux_tl
3771         {
3772             \seq_gset_from_clist:Nn \exp_not:N \g_@@_pos_of_blocks_seq
3773             { \seq_use:Nnnn \g_@@_pos_of_blocks_seq , , , }
3774         }
3775     }
3776 \seq_if_empty:NF \g_@@_multicolumn_cells_seq
3777 {
3778     \tl_gput_right:Nx \g_@@_aux_tl
3779     {
3780         \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_cells_seq
3781         { \seq_use:Nnnn \g_@@_multicolumn_cells_seq , , , }
3782         \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_sizes_seq
3783         { \seq_use:Nnnn \g_@@_multicolumn_sizes_seq , , , }
3784     }
3785 }

```

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

```

3786     \@@_create_diag_nodes:

```

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

```

3787 \pgfpicture
3788 \int_step_inline:nn \c@iRow
3789 {
3790   \pgfnodealias
3791   { \@@_env: - ##1 - last }
3792   { \@@_env: - ##1 - \int_use:N \c@jCol }
3793 }
3794 \int_step_inline:nn \c@jCol
3795 {
3796   \pgfnodealias
3797   { \@@_env: - last - ##1 }
3798   { \@@_env: - \int_use:N \c@iRow - ##1 }
3799 }
3800 \str_if_empty:NF \l_@@_name_str
3801 {
3802   \int_step_inline:nn \c@iRow
3803   {
3804     \pgfnodealias
3805     { \l_@@_name_str - ##1 - last }
3806     { \@@_env: - ##1 - \int_use:N \c@jCol }
3807   }
3808   \int_step_inline:nn \c@jCol
3809   {
3810     \pgfnodealias
3811     { \l_@@_name_str - last - ##1 }
3812     { \@@_env: - \int_use:N \c@iRow - ##1 }
3813   }
3814 }
3815 \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.

```

3816 \bool_if:NT \l_@@_parallelize_diags_bool
3817 {
3818   \int_gzero_new:N \g_@@_ddots_int
3819   \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.

```

3820   \dim_gzero_new:N \g_@@_delta_x_one_dim
3821   \dim_gzero_new:N \g_@@_delta_y_one_dim
3822   \dim_gzero_new:N \g_@@_delta_x_two_dim
3823   \dim_gzero_new:N \g_@@_delta_y_two_dim
3824 }
3825 \int_zero_new:N \l_@@_initial_i_int
3826 \int_zero_new:N \l_@@_initial_j_int
3827 \int_zero_new:N \l_@@_final_i_int
3828 \int_zero_new:N \l_@@_final_j_int
3829 \bool_set_false:N \l_@@_initial_open_bool
3830 \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.

```

3831 \bool_if:NT \l_@@_small_bool
3832 {

```

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

```

3833 \dim_set:Nn \l_@@_xdots_radius_dim { 0.7 \l_@@_xdots_radius_dim }
3834 \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.

```

3835 \dim_set:Nn \l_@@_xdots_shorten_start_dim
3836 { 0.6 \l_@@_xdots_shorten_start_dim }
3837 \dim_set:Nn \l_@@_xdots_shorten_end_dim
3838 { 0.6 \l_@@_xdots_shorten_end_dim }
3839 }

```

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

```

3840 \@@_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.

```

3841 \@@_compute_corners:

```

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

```

3842 \@@_adjust_pos_of_blocks_seq:
3843 \@@_deal_with_rounded_corners:
3844 \tl_if_empty:NF \l_@@_hlines_clist \@@_draw_hlines:
3845 \tl_if_empty:NF \l_@@_vlines_clist \@@_draw_vlines:

```

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

```

3846 \IfPackageLoadedTF { tikz }
3847 {
3848   \tikzset
3849   {
3850     every-picture / .style =
3851     {
3852       overlay ,
3853       remember~picture ,
3854       name~prefix = \@@_env: -
3855     }
3856   }
3857 }
3858 { }
3859 \cs_set_eq:NN \ialign \@@_old_ialign:
3860 \cs_set_eq:NN \SubMatrix \@@_SubMatrix
3861 \cs_set_eq:NN \UnderBrace \@@_UnderBrace
3862 \cs_set_eq:NN \OverBrace \@@_OverBrace
3863 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
3864 \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
3865 \cs_set_eq:NN \line \@@_line
3866 \g_@@_pre_code_after_tl
3867 \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.

```

3868 \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).

```

3869 \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`).

```
3870 \int_compare:nNt { \char_value_catcode:n { 60 } } = { 13 }
3871 { \@@_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:`.

```
3872 \bool_set_true:N \l_@@_in_code_after_bool
3873 \exp_last_unbraced:No \@@_CodeAfter_keys: \g_nicematrix_code_after_tl
3874 \scan_stop:
3875 \tl_gclear:N \g_nicematrix_code_after_tl
3876 \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.

```
3877 \seq_if_empty:NF \g_@@_rowlistcolors_seq { \@@_clear_rowlistcolors_seq: }
3878 \tl_if_empty:NF \g_@@_pre_code_before_tl
3879 {
3880   \tl_gput_right:Nx \g_@@_aux_tl
3881   {
3882     \tl_gset:Nn \exp_not:N \g_@@_pre_code_before_tl
3883     { \exp_not:o \g_@@_pre_code_before_tl }
3884   }
3885   \tl_gclear:N \g_@@_pre_code_before_tl
3886 }
3887 \tl_if_empty:NF \g_nicematrix_code_before_tl
3888 {
3889   \tl_gput_right:Nx \g_@@_aux_tl
3890   {
3891     \tl_gset:Nn \exp_not:N \g_@@_code_before_tl
3892     { \exp_not:o \g_nicematrix_code_before_tl }
3893   }
3894   \tl_gclear:N \g_nicematrix_code_before_tl
3895 }

3896 \str_gclear:N \g_@@_name_env_str
3897 \@@_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`.

```
3898 \cs_gset_eq:NN \CT@arc@ \@@_old_CT@arc@
3899 }
```

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`.

```
3900 \NewDocumentCommand \@@_CodeAfter_keys: { 0 { } }
3901 { \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

¹²e.g. `\color[rgb]{0.5,0.5,0}`

`\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).

```

3902 \cs_new_protected:Npn \@@_adjust_pos_of_blocks_seq:
3903 {
3904   \seq_gset_map_x:NNn \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq
3905   { \@@_adjust_pos_of_blocks_seq_i:nnnnn ##1 }
3906 }

```

The following command must *not* be protected.

```

3907 \cs_new:Npn \@@_adjust_pos_of_blocks_seq_i:nnnnn #1 #2 #3 #4 #5
3908 {
3909   { #1 }
3910   { #2 }
3911   {
3912     \int_compare:nNnTF { #3 } > { 99 }
3913     { \int_use:N \c@iRow }
3914     { #3 }
3915   }
3916   {
3917     \int_compare:nNnTF { #4 } > { 99 }
3918     { \int_use:N \c@jCol }
3919     { #4 }
3920   }
3921   { #5 }
3922 }

```

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).

```

3923 \hook_gput_code:nnn { begindocument } { . }
3924 {
3925   \cs_new_protected:Npx \@@_draw_dotted_lines:
3926   {
3927     \c_@@_pgfortikzpicture_tl
3928     \@@_draw_dotted_lines_i:
3929     \c_@@_endpgfortikzpicture_tl
3930   }
3931 }

```

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

```

3932 \cs_new_protected:Npn \@@_draw_dotted_lines_i:
3933 {
3934   \pgfrememberpicturepositiononpagetrue
3935   \pgf@relevantforpicturesizefalse
3936   \g_@@_HVdotsfor_lines_tl
3937   \g_@@_Vdots_lines_tl
3938   \g_@@_Ddots_lines_tl
3939   \g_@@_Iddots_lines_tl
3940   \g_@@_Cdots_lines_tl
3941   \g_@@_Ldots_lines_tl
3942 }

3943 \cs_new_protected:Npn \@@_restore_iRow_jCol:
3944 {
3945   \cs_if_exist:NT \theiRow { \int_gset_eq:NN \c@iRow \l_@@_old_iRow_int }
3946   \cs_if_exist:NT \thejCol { \int_gset_eq:NN \c@jCol \l_@@_old_jCol_int }
3947 }

```

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

```

3948 \pgfdeclareshape { @@_diag_node }
3949 {
3950   \savedanchor { \five }
3951   {
3952     \dim_gset_eq:NN \pgf@x \l_tmpa_dim
3953     \dim_gset_eq:NN \pgf@y \l_tmpb_dim
3954   }
3955   \anchor { 5 } { \five }
3956   \anchor { center } { \pgfpointorigin }
3957 }

```

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).

```

3958 \cs_new_protected:Npn \@@_create_diag_nodes:
3959 {
3960   \pgfpicture
3961   \pgfrememberpicturepositiononpagetrue
3962   \int_step_inline:nn { \int_max:nn \c@iRow \c@jCol }
3963   {
3964     \@@_qpoint:n { col - \int_min:nn { ##1 } { \c@jCol + 1 } }
3965     \dim_set_eq:NN \l_tmpa_dim \pgf@x
3966     \@@_qpoint:n { row - \int_min:nn { ##1 } { \c@iRow + 1 } }
3967     \dim_set_eq:NN \l_tmpb_dim \pgf@y
3968     \@@_qpoint:n { col - \int_min:nn { ##1 + 1 } { \c@jCol + 1 } }
3969     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
3970     \@@_qpoint:n { row - \int_min:nn { ##1 + 1 } { \c@iRow + 1 } }
3971     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
3972     \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.

```

3973     \dim_set:Nn \l_tmpa_dim { ( \l_@@_tmpc_dim - \l_tmpa_dim ) / 2 }
3974     \dim_set:Nn \l_tmpb_dim { ( \l_@@_tmpd_dim - \l_tmpb_dim ) / 2 }
3975     \pgfnode { @@_diag_node } { center } { } { \@@_env: - ##1 } { }
3976     \str_if_empty:NF \l_@@_name_str
3977     { \pgfnodealias { \l_@@_name_str - ##1 } { \@@_env: - ##1 } }
3978   }

```

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

```

3979   \int_set:Nn \l_tmpa_int { \int_max:nn \c@iRow \c@jCol + 1 }
3980   \@@_qpoint:n { row - \int_min:nn { \l_tmpa_int } { \c@iRow + 1 } }
3981   \dim_set_eq:NN \l_tmpa_dim \pgf@y
3982   \@@_qpoint:n { col - \int_min:nn { \l_tmpa_int } { \c@jCol + 1 } }
3983   \pgfcoordinate
3984   { \@@_env: - \int_use:N \l_tmpa_int } { \pgfpoint \pgf@x \l_tmpa_dim }
3985   \pgfnodealias
3986   { \@@_env: - last }
3987   { \@@_env: - \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
3988   \str_if_empty:NF \l_@@_name_str
3989   {
3990     \pgfnodealias
3991     { \l_@@_name_str - \int_use:N \l_tmpa_int }
3992     { \@@_env: - \int_use:N \l_tmpa_int }
3993     \pgfnodealias
3994     { \l_@@_name_str - last }
3995     { \@@_env: - last }
3996   }
3997   \endpgfpicture
3998 }

```

17 We draw the dotted lines

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

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

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

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

This command computes:

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

```
3999 \cs_new_protected:Npn \@@_find_extremities_of_line:nnnn #1 #2 #3 #4
4000 {
```

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

```
4001 \cs_set:cpn { @@ _ dotted _ #1 - #2 } { }
```

Initialization of variables.

```
4002 \int_set:Nn \l_@@_initial_i_int { #1 }
4003 \int_set:Nn \l_@@_initial_j_int { #2 }
4004 \int_set:Nn \l_@@_final_i_int { #1 }
4005 \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.

```
4006 \bool_set_false:N \l_@@_stop_loop_bool
4007 \bool_do_until:Nn \l_@@_stop_loop_bool
4008 {
4009   \int_add:Nn \l_@@_final_i_int { #3 }
4010   \int_add:Nn \l_@@_final_j_int { #4 }
```

We test if we are still in the matrix.

```
4011 \bool_set_false:N \l_@@_final_open_bool
4012 \int_compare:nNnTF \l_@@_final_i_int > \l_@@_row_max_int
4013 {
4014   \int_compare:nNnTF { #3 } = \c_one_int
4015   { \bool_set_true:N \l_@@_final_open_bool }
4016   {
4017     \int_compare:nNnTF \l_@@_final_j_int > \l_@@_col_max_int
4018     { \bool_set_true:N \l_@@_final_open_bool }
4019   }
4020 }
4021 {
4022   \int_compare:nNnTF \l_@@_final_j_int < \l_@@_col_min_int
```

```

4023     {
4024         \int_compare:nNnT { #4 } = { -1 }
4025         { \bool_set_true:N \l_@@_final_open_bool }
4026     }
4027     {
4028         \int_compare:nNnT \l_@@_final_j_int > \l_@@_col_max_int
4029         {
4030             \int_compare:nNnT { #4 } = \c_one_int
4031             { \bool_set_true:N \l_@@_final_open_bool }
4032         }
4033     }
4034 }
4035 \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.

```

4036     {

```

We do a step backwards.

```

4037         \int_sub:Nn \l_@@_final_i_int { #3 }
4038         \int_sub:Nn \l_@@_final_j_int { #4 }
4039         \bool_set_true:N \l_@@_stop_loop_bool
4040     }

```

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`.

```

4041     {
4042         \cs_if_exist:cTF
4043         {
4044             @@ _ dotted _
4045             \int_use:N \l_@@_final_i_int -
4046             \int_use:N \l_@@_final_j_int
4047         }
4048         {
4049             \int_sub:Nn \l_@@_final_i_int { #3 }
4050             \int_sub:Nn \l_@@_final_j_int { #4 }
4051             \bool_set_true:N \l_@@_final_open_bool
4052             \bool_set_true:N \l_@@_stop_loop_bool
4053         }
4054         {
4055             \cs_if_exist:cTF
4056             {
4057                 pgf @ sh @ ns @ \@@_env:
4058                 - \int_use:N \l_@@_final_i_int
4059                 - \int_use:N \l_@@_final_j_int
4060             }
4061             { \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.

```

4062         {
4063             \cs_set:cpn
4064             {
4065                 @@ _ dotted _
4066                 \int_use:N \l_@@_final_i_int -
4067                 \int_use:N \l_@@_final_j_int
4068             }
4069             { }
4070         }
4071     }
4072 }
4073 }

```

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

```

4074 \bool_set_false:N \l_@@_stop_loop_bool
4075 \bool_do_until:Nn \l_@@_stop_loop_bool
4076 {
4077   \int_sub:Nn \l_@@_initial_i_int { #3 }
4078   \int_sub:Nn \l_@@_initial_j_int { #4 }
4079   \bool_set_false:N \l_@@_initial_open_bool
4080   \int_compare:nNnTF \l_@@_initial_i_int < \l_@@_row_min_int
4081   {
4082     \int_compare:nNnTF { #3 } = \c_one_int
4083     { \bool_set_true:N \l_@@_initial_open_bool }
4084     {
4085       \int_compare:nNnT \l_@@_initial_j_int = { \l_@@_col_min_int - 1 }
4086       { \bool_set_true:N \l_@@_initial_open_bool }
4087     }
4088   }
4089   {
4090     \int_compare:nNnTF \l_@@_initial_j_int < \l_@@_col_min_int
4091     {
4092       \int_compare:nNnT { #4 } = \c_one_int
4093       { \bool_set_true:N \l_@@_initial_open_bool }
4094     }
4095     {
4096       \int_compare:nNnT \l_@@_initial_j_int > \l_@@_col_max_int
4097       {
4098         \int_compare:nNnT { #4 } = { -1 }
4099         { \bool_set_true:N \l_@@_initial_open_bool }
4100       }
4101     }
4102   }
4103   \bool_if:NTF \l_@@_initial_open_bool
4104   {
4105     \int_add:Nn \l_@@_initial_i_int { #3 }
4106     \int_add:Nn \l_@@_initial_j_int { #4 }
4107     \bool_set_true:N \l_@@_stop_loop_bool
4108   }
4109   {
4110     \cs_if_exist:cTF
4111     {
4112       @@ _ dotted _
4113       \int_use:N \l_@@_initial_i_int -
4114       \int_use:N \l_@@_initial_j_int
4115     }
4116     {
4117       \int_add:Nn \l_@@_initial_i_int { #3 }
4118       \int_add:Nn \l_@@_initial_j_int { #4 }
4119       \bool_set_true:N \l_@@_initial_open_bool
4120       \bool_set_true:N \l_@@_stop_loop_bool
4121     }
4122     {
4123       \cs_if_exist:cTF
4124       {
4125         pgf @ sh @ ns @ \@@_env:
4126         - \int_use:N \l_@@_initial_i_int
4127         - \int_use:N \l_@@_initial_j_int
4128       }
4129       { \bool_set_true:N \l_@@_stop_loop_bool }
4130       {
4131         \cs_set:cpn
4132         {
4133           @@ _ dotted _
4134           \int_use:N \l_@@_initial_i_int -

```

```

4135             \int_use:N \l_@@_initial_j_int
4136         }
4137     { }
4138 }
4139 }
4140 }
4141 }

```

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.

```

4142 \seq_gput_right:Nx \g_@@_pos_of_xdots_seq
4143 {
4144     { \int_use:N \l_@@_initial_i_int }

```

Be careful: with `\Iddots`, `\l_@@_final_j_int` is inferior to `\l_@@_initial_j_int`. That’s why we use `\int_min:nn` and `\int_max:nn`.

```

4145     { \int_min:nn \l_@@_initial_j_int \l_@@_final_j_int }
4146     { \int_use:N \l_@@_final_i_int }
4147     { \int_max:nn \l_@@_initial_j_int \l_@@_final_j_int }
4148     { } % for the name of the block
4149 }
4150 }

```

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

```

4151 \cs_new_protected:Npn \@@_open_shorten:
4152 {
4153     \bool_if:NT \l_@@_initial_open_bool
4154     { \dim_zero:N \l_@@_xdots_shorten_start_dim }
4155     \bool_if:NT \l_@@_final_open_bool
4156     { \dim_zero:N \l_@@_xdots_shorten_end_dim }
4157 }

```

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).

```

4158 \cs_new_protected:Npn \@@_adjust_to_submatrix:nn #1 #2
4159 {
4160     \int_set:Nn \l_@@_row_min_int 1
4161     \int_set:Nn \l_@@_col_min_int 1
4162     \int_set_eq:NN \l_@@_row_max_int \c@iRow
4163     \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`.

```

4164 \seq_map_inline:Nn \g_@@_submatrix_seq
4165 { \@@_adjust_to_submatrix:nnnnnn { #1 } { #2 } ##1 }
4166 }

```

#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.

```

4167 \cs_set_protected:Npn \@@_adjust_to_submatrix:nnnnnn #1 #2 #3 #4 #5 #6
4168 {
4169     \int_compare:nNnF { #3 } > { #1 }
4170     {
4171         \int_compare:nNnF { #1 } > { #5 }
4172         {

```

```

4173         \int_compare:nNnF { #4 } > { #2 }
4174         {
4175             \int_compare:nNnF { #2 } > { #6 }
4176             {
4177                 \int_set:Nn \l_@@_row_min_int
4178                 { \int_max:nn \l_@@_row_min_int { #3 } }
4179                 \int_set:Nn \l_@@_col_min_int
4180                 { \int_max:nn \l_@@_col_min_int { #4 } }
4181                 \int_set:Nn \l_@@_row_max_int
4182                 { \int_min:nn \l_@@_row_max_int { #5 } }
4183                 \int_set:Nn \l_@@_col_max_int
4184                 { \int_min:nn \l_@@_col_max_int { #6 } }
4185             }
4186         }
4187     }
4188 }
4189 }

4190 \cs_new_protected:Npn \@@_set_initial_coords:
4191 {
4192     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4193     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4194 }
4195 \cs_new_protected:Npn \@@_set_final_coords:
4196 {
4197     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4198     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4199 }
4200 \cs_new_protected:Npn \@@_set_initial_coords_from_anchor:n #1
4201 {
4202     \pgfpointanchor
4203     {
4204         \@@_env:
4205         - \int_use:N \l_@@_initial_i_int
4206         - \int_use:N \l_@@_initial_j_int
4207     }
4208     { #1 }
4209     \@@_set_initial_coords:
4210 }
4211 \cs_new_protected:Npn \@@_set_final_coords_from_anchor:n #1
4212 {
4213     \pgfpointanchor
4214     {
4215         \@@_env:
4216         - \int_use:N \l_@@_final_i_int
4217         - \int_use:N \l_@@_final_j_int
4218     }
4219     { #1 }
4220     \@@_set_final_coords:
4221 }
4222 \cs_new_protected:Npn \@@_open_x_initial_dim:
4223 {
4224     \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
4225     \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
4226     {
4227         \cs_if_exist:cT
4228         { \pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4229         {
4230             \pgfpointanchor
4231             { \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4232             { west }
4233             \dim_set:Nn \l_@@_x_initial_dim
4234             { \dim_min:nn \l_@@_x_initial_dim \pgf@x }

```

```

4235     }
4236 }

```

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

```

4237 \dim_compare:nNt \l_@@_x_initial_dim = \c_max_dim
4238 {
4239   \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4240   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4241   \dim_add:Nn \l_@@_x_initial_dim \col@sep
4242 }
4243 }
4244 \cs_new_protected:Npn \@@_open_x_final_dim:
4245 {
4246   \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
4247   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
4248   {
4249     \cs_if_exist:cT
4250     { \pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4251     {
4252       \pgfpoinanchor
4253       { \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4254       { east }
4255       \dim_set:Nn \l_@@_x_final_dim
4256       { \dim_max:nn \l_@@_x_final_dim \pgf@x }
4257     }
4258   }

```

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

```

4259 \dim_compare:nNt \l_@@_x_final_dim = { - \c_max_dim }
4260 {
4261   \@@_qpoint:n { col - \int_eval:n { \l_@@_final_j_int + 1 } }
4262   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4263   \dim_sub:Nn \l_@@_x_final_dim \col@sep
4264 }
4265 }

```

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.

```

4266 \cs_new_protected:Npn \@@_draw_Ldots:nnn #1 #2 #3
4267 {
4268   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4269   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4270   {
4271     \@@_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.

```

4272 \group_begin:
4273 \@@_open_shorten:
4274 \int_if_zero:nTF { #1 }
4275 { \color { nicematrix-first-row } }
4276 {

```

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.

```

4277 \int_compare:nNt { #1 } = \l_@@_last_row_int
4278 { \color { nicematrix-last-row } }
4279 }
4280 \keys_set:nn { NiceMatrix / xdots } { #3 }
4281 \tl_if_empty:oF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4282 \@@_actually_draw_Ldots:
4283 \group_end:
4284 }
4285 }

```

The command `\@@_actually_draw_Ldots:` has the following implicit arguments:

- `\l_@@_initial_i_int`
- `\l_@@_initial_j_int`
- `\l_@@_initial_open_bool`
- `\l_@@_final_i_int`
- `\l_@@_final_j_int`
- `\l_@@_final_open_bool.`

The following function is also used by `\Hdotsfor`.

```

4286 \cs_new_protected:Npn \@@_actually_draw_Ldots:
4287 {
4288   \bool_if:NTF \l_@@_initial_open_bool
4289   {
4290     \@@_open_x_initial_dim:
4291     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4292     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4293   }
4294   { \@@_set_initial_coords_from_anchor:n { base~east } }
4295   \bool_if:NTF \l_@@_final_open_bool
4296   {
4297     \@@_open_x_final_dim:
4298     \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4299     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4300   }
4301   { \@@_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.

```

4302   \bool_lazy_all:nTF
4303   {
4304     \l_@@_initial_open_bool
4305     \l_@@_final_open_bool
4306     { \int_compare_p:nNn \l_@@_initial_i_int = \l_@@_last_row_int }
4307   }
4308   {
4309     \dim_add:Nn \l_@@_y_initial_dim \c_@@_shift_Ldots_last_row_dim
4310     \dim_add:Nn \l_@@_y_final_dim \c_@@_shift_Ldots_last_row_dim
4311   }

```

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 (?).

```

4312   {
4313     \dim_add:Nn \l_@@_y_initial_dim \l_@@_xdots_radius_dim
4314     \dim_add:Nn \l_@@_y_final_dim \l_@@_xdots_radius_dim
4315   }
4316   \@@_draw_line:
4317 }

```

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.

```

4318 \cs_new_protected:Npn \@@_draw_Cdots:nnn #1 #2 #3
4319 {
4320   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4321   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4322   {
4323     \@@_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.

```

4324 \group_begin:
4325 \@@_open_shorten:
4326 \int_if_zero:nTF { #1 }
4327 { \color { nicematrix-first-row } }
4328 {

```

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.

```

4329 \int_compare:nNnT { #1 } = \l_@@_last_row_int
4330 { \color { nicematrix-last-row } }
4331 }
4332 \keys_set:nn { NiceMatrix / xdots } { #3 }
4333 \tl_if_empty:oF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4334 \@@_actually_draw_Cdots:
4335 \group_end:
4336 }
4337 }

```

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`.

```

4338 \cs_new_protected:Npn \@@_actually_draw_Cdots:
4339 {
4340 \bool_if:NTF \l_@@_initial_open_bool
4341 { \@@_open_x_initial_dim: }
4342 { \@@_set_initial_coords_from_anchor:n { mid-east } }
4343 \bool_if:NTF \l_@@_final_open_bool
4344 { \@@_open_x_final_dim: }
4345 { \@@_set_final_coords_from_anchor:n { mid-west } }
4346 \bool_lazy_and:nnTF
4347 \l_@@_initial_open_bool
4348 \l_@@_final_open_bool
4349 {
4350 \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int }
4351 \dim_set_eq:NN \l_tmpa_dim \pgf@y
4352 \@@_qpoint:n { row - \int_eval:n { \l_@@_initial_i_int + 1 } }
4353 \dim_set:Nn \l_@@_y_initial_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
4354 \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
4355 }
4356 {
4357 \bool_if:NT \l_@@_initial_open_bool
4358 { \dim_set_eq:NN \l_@@_y_initial_dim \l_@@_y_final_dim }
4359 \bool_if:NT \l_@@_final_open_bool
4360 { \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim }
4361 }
4362 \@@_draw_line:
4363 }
4364 \cs_new_protected:Npn \@@_open_y_initial_dim:
4365 {
4366 \dim_set:Nn \l_@@_y_initial_dim { - \c_max_dim }
4367 \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
4368 {

```

```

4369 \cs_if_exist:cT
4370 { pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4371 {
4372   \pgfpointanchor
4373   { \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4374   { north }
4375   \dim_set:Nn \l_@@_y_initial_dim
4376   { \dim_max:nn \l_@@_y_initial_dim \pgf@y }
4377 }
4378 }
4379 \dim_compare:nNnT \l_@@_y_initial_dim = { - \c_max_dim }
4380 {
4381   \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4382   \dim_set:Nn \l_@@_y_initial_dim
4383   {
4384     \fp_to_dim:n
4385     {
4386       \pgf@y
4387       + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
4388     }
4389   }
4390 }
4391 }
4392 \cs_new_protected:Npn \@@_open_y_final_dim:
4393 {
4394   \dim_set_eq:NN \l_@@_y_final_dim \c_max_dim
4395   \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
4396   {
4397     \cs_if_exist:cT
4398     { pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4399     {
4400       \pgfpointanchor
4401       { \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4402       { south }
4403       \dim_set:Nn \l_@@_y_final_dim
4404       { \dim_min:nn \l_@@_y_final_dim \pgf@y }
4405     }
4406   }
4407   \dim_compare:nNnT \l_@@_y_final_dim = \c_max_dim
4408   {
4409     \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4410     \dim_set:Nn \l_@@_y_final_dim
4411     { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
4412   }
4413 }

```

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.

```

4414 \cs_new_protected:Npn \@@_draw_Vdots:nnn #1 #2 #3
4415 {
4416   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4417   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4418   {
4419     \@@_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.

```

4420 \group_begin:
4421 \@@_open_shorten:
4422 \int_if_zero:nTF { #2 }
4423 { \color { nicematrix-first-col } }
4424 {
4425   \int_compare:nNnT { #2 } = \l_@@_last_col_int
4426   { \color { nicematrix-last-col } }

```

```

4427     }
4428     \keys_set:nn { NiceMatrix / xdots } { #3 }
4429     \tl_if_empty:oF \l_@@_xdots_color_tl
4430     { \color { \l_@@_xdots_color_tl } }
4431     \@@_actually_draw_Vdots:
4432     \group_end:
4433   }
4434 }

```

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`.

```

4435 \cs_new_protected:Npn \@@_actually_draw_Vdots:
4436 {

```

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

```

4437   \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.

```

4438   {
4439     \@@_open_y_initial_dim:
4440     \@@_open_y_final_dim:
4441     \int_if_zero:nTF \l_@@_initial_j_int

```

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

```

4442   {
4443     \@@_qpoint:n { col - 1 }
4444     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4445     \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
4446     \dim_sub:Nn \l_@@_x_initial_dim \l_@@_extra_left_margin_dim
4447     \dim_sub:Nn \l_@@_x_initial_dim \c_@@_shift_exterior_Vdots_dim
4448   }
4449   {
4450     \bool_lazy_and:nnTF
4451     { \int_compare_p:nNn \l_@@_last_col_int > { -2 } }
4452     { \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”.

```

4453   {
4454     \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4455     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4456     \dim_add:Nn \l_@@_x_initial_dim \l_@@_right_margin_dim
4457     \dim_add:Nn \l_@@_x_initial_dim \l_@@_extra_right_margin_dim
4458     \dim_add:Nn \l_@@_x_initial_dim \c_@@_shift_exterior_Vdots_dim
4459   }

```

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

```

4460   {
4461     \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4462     \dim_set_eq:NN \l_tmpa_dim \pgf@x
4463     \@@_qpoint:n { col - \int_eval:n { \l_@@_initial_j_int + 1 } }
4464     \dim_set:Nn \l_@@_x_initial_dim { ( \pgf@x + \l_tmpa_dim ) / 2 }
4465   }
4466 }
4467 }

```

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.

```

4468 {
4469     \bool_set_false:N \l_tmpa_bool
4470     \bool_if:NF \l_@@_initial_open_bool
4471     {
4472         \bool_if:NF \l_@@_final_open_bool
4473         {
4474             \@@_set_initial_coords_from_anchor:n { south-west }
4475             \@@_set_final_coords_from_anchor:n { north-west }
4476             \bool_set:Nn \l_tmpa_bool
4477             { \dim_compare_p:nNn \l_@@_x_initial_dim = \l_@@_x_final_dim }
4478         }
4479     }

```

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

```

4480     \bool_if:NTF \l_@@_initial_open_bool
4481     {
4482         \@@_open_y_initial_dim:
4483         \@@_set_final_coords_from_anchor:n { north }
4484         \dim_set_eq:NN \l_@@_x_initial_dim \l_@@_x_final_dim
4485     }
4486     {
4487         \@@_set_initial_coords_from_anchor:n { south }
4488         \bool_if:NTF \l_@@_final_open_bool
4489         \@@_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.

```

4490         {
4491             \@@_set_final_coords_from_anchor:n { north }
4492             \dim_compare:nNnF \l_@@_x_initial_dim = \l_@@_x_final_dim
4493             {
4494                 \dim_set:Nn \l_@@_x_initial_dim
4495                 {
4496                     \bool_if:NTF \l_tmpa_bool \dim_min:nn \dim_max:nn
4497                     \l_@@_x_initial_dim \l_@@_x_final_dim
4498                 }
4499             }
4500         }
4501     }
4502 }
4503 \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
4504 \@@_draw_line:
4505 }

```

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.

```

4506 \cs_new_protected:Npn \@@_draw_Ddots:nnn #1 #2 #3
4507 {
4508     \@@_adjust_to_submatrix:nn { #1 } { #2 }
4509     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4510     {
4511         \@@_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.

```

4512     \group_begin:
4513     \@@_open_shorten:

```

```

4514         \keys_set:nn { NiceMatrix / xdots } { #3 }
4515         \tl_if_empty:oF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4516         \@@_actually_draw_Ddots:
4517     \group_end:
4518 }
4519 }

```

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.`

```

4520 \cs_new_protected:Npn \@@_actually_draw_Ddots:
4521 {
4522     \bool_if:NTF \l_@@_initial_open_bool
4523     {
4524         \@@_open_y_initial_dim:
4525         \@@_open_x_initial_dim:
4526     }
4527     { \@@_set_initial_coords_from_anchor:n { south~east } }
4528     \bool_if:NTF \l_@@_final_open_bool
4529     {
4530         \@@_open_x_final_dim:
4531         \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4532     }
4533     { \@@_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.

```

4534     \bool_if:NT \l_@@_parallelize_diags_bool
4535     {
4536         \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).

```

4537     \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.

```

4538     {
4539         \dim_gset:Nn \g_@@_delta_x_one_dim
4540         { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4541         \dim_gset:Nn \g_@@_delta_y_one_dim
4542         { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4543     }

```

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`.

```

4544     {
4545         \dim_set:Nn \l_@@_y_final_dim
4546         {
4547             \l_@@_y_initial_dim +
4548             ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4549             \dim_ratio:nn \g_@@_delta_y_one_dim \g_@@_delta_x_one_dim
4550         }
4551     }
4552 }
4553 \@@_draw_line:
4554 }

```

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.

```

4555 \cs_new_protected:Npn \@@_draw_Iddots:nnn #1 #2 #3
4556 {
4557   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4558   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4559   {
4560     \@@_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.

```

4561     \group_begin:
4562     \@@_open_shorten:
4563     \keys_set:nn { NiceMatrix / xdots } { #3 }
4564     \tl_if_empty:oF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4565     \@@_actually_draw_Iddots:
4566   \group_end:
4567 }
4568 }

```

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`.

```

4569 \cs_new_protected:Npn \@@_actually_draw_Iddots:
4570 {
4571   \bool_if:NTF \l_@@_initial_open_bool
4572   {
4573     \@@_open_y_initial_dim:
4574     \@@_open_x_initial_dim:
4575   }
4576   { \@@_set_initial_coords_from_anchor:n { south~west } }
4577   \bool_if:NTF \l_@@_final_open_bool
4578   {
4579     \@@_open_y_final_dim:
4580     \@@_open_x_final_dim:
4581   }
4582   { \@@_set_final_coords_from_anchor:n { north~east } }
4583   \bool_if:NT \l_@@_parallelize_diags_bool
4584   {
4585     \int_gincr:N \g_@@_iddots_int
4586     \int_compare:nNnTF \g_@@_iddots_int = \c_one_int
4587     {
4588       \dim_gset:Nn \g_@@_delta_x_two_dim
4589       { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4590       \dim_gset:Nn \g_@@_delta_y_two_dim
4591       { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4592     }
4593     {
4594       \dim_set:Nn \l_@@_y_final_dim
4595       {
4596         \l_@@_y_initial_dim +
4597         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4598         \dim_ratio:nn \g_@@_delta_y_two_dim \g_@@_delta_x_two_dim

```

```

4599         }
4600     }
4601 }
4602 \@@_draw_line:
4603 }

```

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`

```

4604 \cs_new_protected:Npn \@@_draw_line:
4605 {
4606     \pgfrememberpicturepositiononpagetrue
4607     \pgf@relevantforpicturesizefalse
4608     \bool_lazy_or:nnTF
4609     { \tl_if_eq_p:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl }
4610     \l_@@_dotted_bool
4611     \@@_draw_standard_dotted_line:
4612     \@@_draw_unstandard_dotted_line:
4613 }

```

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.

```

4614 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:
4615 {
4616     \begin { scope }
4617     \@@_draw_unstandard_dotted_line:o
4618     { \l_@@_xdots_line_style_tl , \l_@@_xdots_color_tl }
4619 }

```

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.

```

4620 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:n #1
4621 {
4622     \@@_draw_unstandard_dotted_line:nooo
4623     { #1 }
4624     \l_@@_xdots_up_tl
4625     \l_@@_xdots_down_tl
4626     \l_@@_xdots_middle_tl
4627 }
4628 \cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:n { o }

```

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

```

4629 \hook_gput_code:nnn { begindocument } { . }
4630 {
4631   \IfPackageLoadedTF { tikz }
4632   {
4633     \tikzset
4634     {
4635       @@_node_above / .style = { sloped , above } ,
4636       @@_node_below / .style = { sloped , below } ,
4637       @@_node_middle / .style =
4638       {
4639         sloped ,
4640         inner-sep = \c_@@_innersep_middle_dim
4641       }
4642     }
4643   }
4644 }
4645 }

4646 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:nnnn #1 #2 #3 #4
4647 {

```

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.

```

4648 \dim_zero_new:N \l_@@_l_dim
4649 \dim_set:Nn \l_@@_l_dim
4650 {
4651   \fp_to_dim:n
4652   {
4653     sqrt
4654     (
4655       ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
4656       +
4657       ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
4658     )
4659   }
4660 }

```

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.

```

4661 \dim_compare:nNnT \l_@@_l_dim < \c_@@_max_l_dim
4662 {
4663   \dim_compare:nNnT \l_@@_l_dim > { 1 pt }
4664   \@@_draw_unstandard_dotted_line_i:
4665 }

```

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

```

4666 \bool_if:NT \l_@@_xdots_h_labels_bool
4667 {
4668   \tikzset
4669   {
4670     @@_node_above / .style = { auto = left } ,
4671     @@_node_below / .style = { auto = right } ,
4672     @@_node_middle / .style = { inner-sep = \c_@@_innersep_middle_dim }
4673   }
4674 }

```

```

4675 \tl_if_empty:nF { #4 }
4676 { \tikzset { @@_node_middle / .append~style = { fill = white } } }
4677 \draw
4678 [ #1 ]
4679 ( \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).

```

4680 -- node [ @@_node_middle ] { $ \scriptstyle #4 $ }
4681 node [ @@_node_below ] { $ \scriptstyle #3 $ }
4682 node [ @@_node_above ] { $ \scriptstyle #2 $ }
4683 ( \l_@@_x_final_dim , \l_@@_y_final_dim ) ;
4684 \end { scope }
4685 }

4686 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line_i:
4687 {
4688 \dim_set:Nn \l_tmpa_dim
4689 {
4690 \l_@@_x_initial_dim
4691 + ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
4692 * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
4693 }
4694 \dim_set:Nn \l_tmpb_dim
4695 {
4696 \l_@@_y_initial_dim
4697 + ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
4698 * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
4699 }
4700 \dim_set:Nn \l_@@_tmpc_dim
4701 {
4702 \l_@@_x_final_dim
4703 - ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
4704 * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
4705 }
4706 \dim_set:Nn \l_@@_tmpd_dim
4707 {
4708 \l_@@_y_final_dim
4709 - ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
4710 * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
4711 }
4712 \dim_set_eq:NN \l_@@_x_initial_dim \l_tmpa_dim
4713 \dim_set_eq:NN \l_@@_y_initial_dim \l_tmpb_dim
4714 \dim_set_eq:NN \l_@@_x_final_dim \l_@@_tmpc_dim
4715 \dim_set_eq:NN \l_@@_y_final_dim \l_@@_tmpd_dim
4716 }

4717 \cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:nnnn { n o o o }

```

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

```

4718 \cs_new_protected:Npn \@@_draw_standard_dotted_line:
4719 {
4720 \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.

```

4721 \dim_zero_new:N \l_@@_l_dim
4722 \dim_set:Nn \l_@@_l_dim
4723 {
4724 \fp_to_dim:n
4725 {
4726 sqrt
4727 (

```

```

4728         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
4729         +
4730         ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
4731     )
4732 }
4733 }

```

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.

```

4734 \dim_compare:nNnT \l_@@_l_dim < \c_@@_max_l_dim
4735 {
4736     \dim_compare:nNnT \l_@@_l_dim > { 1 pt }
4737     \@@_draw_standard_dotted_line_i:
4738 }
4739 \group_end:
4740 \bool_lazy_all:nF
4741 {
4742     { \tl_if_empty_p:N \l_@@_xdots_up_tl }
4743     { \tl_if_empty_p:N \l_@@_xdots_down_tl }
4744     { \tl_if_empty_p:N \l_@@_xdots_middle_tl }
4745 }
4746 \l_@@_labels_standard_dotted_line:
4747 }
4748 \dim_const:Nn \c_@@_max_l_dim { 50 cm }
4749 \cs_new_protected:Npn \@@_draw_standard_dotted_line_i:
4750 {

```

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

```

4751 \int_set:Nn \l_tmpa_int
4752 {
4753     \dim_ratio:nn
4754     {
4755         \l_@@_l_dim
4756         - \l_@@_xdots_shorten_start_dim
4757         - \l_@@_xdots_shorten_end_dim
4758     }
4759     \l_@@_xdots_inter_dim
4760 }

```

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

```

4761 \dim_set:Nn \l_tmpa_dim
4762 {
4763     ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4764     \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
4765 }
4766 \dim_set:Nn \l_tmpb_dim
4767 {
4768     ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
4769     \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
4770 }

```

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.

```

4771 \dim_gadd:Nn \l_@@_x_initial_dim
4772 {
4773     ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4774     \dim_ratio:nn
4775     {
4776         \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int

```

```

4777         + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
4778     }
4779     { 2 \l_@@_l_dim }
4780 }
4781 \dim_gadd:Nn \l_@@_y_initial_dim
4782 {
4783     ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
4784     \dim_ratio:nn
4785     {
4786         \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
4787         + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
4788     }
4789     { 2 \l_@@_l_dim }
4790 }
4791 \pgf@relevantforpicturesizefalse
4792 \int_step_inline:nnn \c_zero_int \l_tmpa_int
4793 {
4794     \pgfpathcircle
4795     { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
4796     { \l_@@_xdots_radius_dim }
4797     \dim_add:Nn \l_@@_x_initial_dim \l_tmpa_dim
4798     \dim_add:Nn \l_@@_y_initial_dim \l_tmpb_dim
4799 }
4800 \pgfusepathqfill
4801 }

4802 \cs_new_protected:Npn \l_@@_labels_standard_dotted_line:
4803 {
4804     \pgfscope
4805     \pgftransformshift
4806     {
4807         \pgfpointlineattime { 0.5 }
4808         { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
4809         { \pgfpoint \l_@@_x_final_dim \l_@@_y_final_dim }
4810     }
4811     \fp_set:Nn \l_tmpa_fp
4812     {
4813         atand
4814         (
4815             \l_@@_y_final_dim - \l_@@_y_initial_dim ,
4816             \l_@@_x_final_dim - \l_@@_x_initial_dim
4817         )
4818     }
4819     \pgftransformrotate { \fp_use:N \l_tmpa_fp }
4820     \bool_if:NF \l_@@_xdots_h_labels_bool { \fp_zero:N \l_tmpa_fp }
4821     \tl_if_empty:NF \l_@@_xdots_middle_tl
4822     {
4823         \begin { pgfscope }
4824         \pgfset { inner~sep = \c_@@_innersep_middle_dim }
4825         \pgfnode
4826         { rectangle }
4827         { center }
4828         {
4829             \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4830             {
4831                 \c_math_toggle_token
4832                 \scriptstyle \l_@@_xdots_middle_tl
4833                 \c_math_toggle_token
4834             }
4835         }
4836         { }
4837         {
4838             \pgfsetfillcolor { white }

```

```

4839         \pgfusepath { fill }
4840     }
4841     \end { pgfscope }
4842 }
4843 \tl_if_empty:NF \l_@@_xdots_up_tl
4844 {
4845     \pgfnode
4846     { rectangle }
4847     { south }
4848     {
4849         \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4850         {
4851             \c_math_toggle_token
4852             \scriptstyle \l_@@_xdots_up_tl
4853             \c_math_toggle_token
4854         }
4855     }
4856     { }
4857     { \pgfusepath { } }
4858 }
4859 \tl_if_empty:NF \l_@@_xdots_down_tl
4860 {
4861     \pgfnode
4862     { rectangle }
4863     { north }
4864     {
4865         \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4866         {
4867             \c_math_toggle_token
4868             \scriptstyle \l_@@_xdots_down_tl
4869             \c_math_toggle_token
4870         }
4871     }
4872     { }
4873     { \pgfusepath { } }
4874 }
4875 \endpgfscope
4876 }

```

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.

```

4877 \hook_gput_code:nnn { begindocument } { . }
4878 {
4879     \cs_set_nopar:Npn \l_@@_argspec_tl { m E { _ ^ : } { { } { } { } } }
4880     \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
4881     \cs_new_protected:Npn \@@_Ldots
4882     { \@@_collect_options:n { \@@_Ldots_i } }
4883     \exp_args:NNo \NewDocumentCommand \@@_Ldots_i \l_@@_argspec_tl
4884     {
4885         \int_if_zero:nTF \c@jCol

```

```

4886 { \@@_error:nn { in~first~col } \Ldots }
4887 {
4888   \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
4889   { \@@_error:nn { in~last~col } \Ldots }
4890   {
4891     \@@_instruction_of_type:nnn \c_false_bool { Ldots }
4892     { #1 , down = #2 , up = #3 , middle = #4 }
4893   }
4894 }
4895 \bool_if:NF \l_@@_nullify_dots_bool
4896 { \phantom { \ensuremath { \@@_old_ldots } } }
4897 \bool_gset_true:N \g_@@_empty_cell_bool
4898 }

4899 \cs_new_protected:Npn \@@_Cdots
4900 { \@@_collect_options:n { \@@_Cdots_i } }
4901 \exp_args:NNo \NewDocumentCommand \@@_Cdots_i \l_@@_argspec_tl
4902 {
4903   \int_if_zero:nTF \c@jCol
4904   { \@@_error:nn { in~first~col } \Cdots }
4905   {
4906     \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
4907     { \@@_error:nn { in~last~col } \Cdots }
4908     {
4909       \@@_instruction_of_type:nnn \c_false_bool { Cdots }
4910       { #1 , down = #2 , up = #3 , middle = #4 }
4911     }
4912   }
4913   \bool_if:NF \l_@@_nullify_dots_bool
4914   { \phantom { \ensuremath { \@@_old_cdots } } }
4915   \bool_gset_true:N \g_@@_empty_cell_bool
4916 }

4917 \cs_new_protected:Npn \@@_Vdots
4918 { \@@_collect_options:n { \@@_Vdots_i } }
4919 \exp_args:NNo \NewDocumentCommand \@@_Vdots_i \l_@@_argspec_tl
4920 {
4921   \int_if_zero:nTF \c@iRow
4922   { \@@_error:nn { in~first~row } \Vdots }
4923   {
4924     \int_compare:nNnTF \c@iRow = \l_@@_last_row_int
4925     { \@@_error:nn { in~last~row } \Vdots }
4926     {
4927       \@@_instruction_of_type:nnn \c_false_bool { Vdots }
4928       { #1 , down = #2 , up = #3 , middle = #4 }
4929     }
4930   }
4931   \bool_if:NF \l_@@_nullify_dots_bool
4932   { \phantom { \ensuremath { \@@_old_vdots } } }
4933   \bool_gset_true:N \g_@@_empty_cell_bool
4934 }

4935 \cs_new_protected:Npn \@@_Ddots
4936 { \@@_collect_options:n { \@@_Ddots_i } }
4937 \exp_args:NNo \NewDocumentCommand \@@_Ddots_i \l_@@_argspec_tl
4938 {
4939   \int_case:nnF \c@iRow
4940   {
4941     0 { \@@_error:nn { in~first~row } \Ddots }
4942     \l_@@_last_row_int { \@@_error:nn { in~last~row } \Ddots }
4943   }

```

```

4944 {
4945   \int_case:nnF \c@jCol
4946   {
4947     0 { \@@_error:nn { in~first~col } \Ddots }
4948     \l_@@_last_col_int { \@@_error:nn { in~last~col } \Ddots }
4949   }
4950   {
4951     \keys_set_known:nn { NiceMatrix / Ddots } { #1 }
4952     \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Ddots }
4953     { #1 , down = #2 , up = #3 , middle = #4 }
4954   }
4955 }
4956 }
4957 \bool_if:NF \l_@@_nullify_dots_bool
4958 { \phantom { \ensuremath { \@@_old_ddots } } }
4959 \bool_gset_true:N \g_@@_empty_cell_bool
4960 }

4961 \cs_new_protected:Npn \@@_Iddots
4962 { \@@_collect_options:n { \@@_Iddots_i } }
4963 \exp_args:NNo \NewDocumentCommand \@@_Iddots_i \l_@@_argspec_tl
4964 {
4965   \int_case:nnF \c@iRow
4966   {
4967     0 { \@@_error:nn { in~first~row } \Iddots }
4968     \l_@@_last_row_int { \@@_error:nn { in~last~row } \Iddots }
4969   }
4970   {
4971     \int_case:nnF \c@jCol
4972     {
4973       0 { \@@_error:nn { in~first~col } \Iddots }
4974       \l_@@_last_col_int { \@@_error:nn { in~last~col } \Iddots }
4975     }
4976     {
4977       \keys_set_known:nn { NiceMatrix / Ddots } { #1 }
4978       \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Iddots }
4979       { #1 , down = #2 , up = #3 , middle = #4 }
4980     }
4981   }
4982   \bool_if:NF \l_@@_nullify_dots_bool
4983   { \phantom { \ensuremath { \@@_old_iddots } } }
4984   \bool_gset_true:N \g_@@_empty_cell_bool
4985 }
4986 }

```

End of the \AddToHook.

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

```

4987 \keys_define:nn { NiceMatrix / Ddots }
4988 {
4989   draw-first .bool_set:N = \l_@@_draw_first_bool ,
4990   draw-first .default:n = true ,
4991   draw-first .value_forbidden:n = true
4992 }

```

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

```

4993 \cs_new_protected:Npn \@@_Hspace:
4994 {
4995   \bool_gset_true:N \g_@@_empty_cell_bool
4996   \hspace
4997 }

```

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`.

```
4998 \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`.

```
4999 \cs_new:Npn \@_Hdotsfor:
5000 {
5001   \bool_lazy_and:nnTF
5002     { \int_if_zero_p:n \c@jCol }
5003     { \int_if_zero_p:n \l_@@_first_col_int }
5004     {
5005       \bool_if:NTF \g_@@_after_col_zero_bool
5006       {
5007         \multicolumn { 1 } { c } { }
5008         \@_Hdotsfor_i
5009       }
5010       { \@_fatal:n { Hdotsfor~in~col~0 } }
5011     }
5012   {
5013     \multicolumn { 1 } { c } { }
5014     \@_Hdotsfor_i
5015   }
5016 }
```

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:`).

```
5017 \hook_gput_code:nnn { begindocument } { . }
5018 {
5019   \cs_set_nopar:Npn \l_@@_argspec_tl { m m 0 { } E { _ ^ : } { { } { } { } } }
5020   \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.

```
5021   \cs_new_protected:Npn \@_Hdotsfor_i
5022     { \@_collect_options:n { \@_Hdotsfor_ii } }
5023   \exp_args:NNo \NewDocumentCommand \@_Hdotsfor_ii \l_@@_argspec_tl
5024     {
5025       \tl_gput_right:Nx \g_@@_HVDotsfor_lines_tl
5026       {
5027         \@_Hdotsfor:nnnn
5028         { \int_use:N \c@iRow }
5029         { \int_use:N \c@jCol }
5030         { #2 }
5031         {
5032           #1 , #3 ,
5033           down = \exp_not:n { #4 } ,
5034           up = \exp_not:n { #5 } ,
5035           middle = \exp_not:n { #6 }
5036         }
5037       }
5038       \prg_replicate:nn { #2 - 1 }
5039       {
5040         &
5041         \multicolumn { 1 } { c } { }
5042         \cs_set_eq:NN \CodeAfter \@_CodeAfter_i:
5043       }
5044     }
5045 }
```

```

5046 \cs_new_protected:Npn \@@_Hdotsfor:nnnn #1 #2 #3 #4
5047 {
5048   \bool_set_false:N \l_@@_initial_open_bool
5049   \bool_set_false:N \l_@@_final_open_bool

```

For the row, it's easy.

```

5050   \int_set:Nn \l_@@_initial_i_int { #1 }
5051   \int_set_eq:NN \l_@@_final_i_int \l_@@_initial_i_int

```

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

```

5052   \int_compare:nNnTF { #2 } = \c_one_int
5053   {
5054     \int_set_eq:NN \l_@@_initial_j_int \c_one_int
5055     \bool_set_true:N \l_@@_initial_open_bool
5056   }
5057   {
5058     \cs_if_exist:cTF
5059     {
5060       pgf @ sh @ ns @ \@@_env:
5061       - \int_use:N \l_@@_initial_i_int
5062       - \int_eval:n { #2 - 1 }
5063     }
5064     { \int_set:Nn \l_@@_initial_j_int { #2 - 1 } }
5065     {
5066       \int_set:Nn \l_@@_initial_j_int { #2 }
5067       \bool_set_true:N \l_@@_initial_open_bool
5068     }
5069   }
5070   \int_compare:nNnTF { #2 + #3 - 1 } = \c_jCol
5071   {
5072     \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5073     \bool_set_true:N \l_@@_final_open_bool
5074   }
5075   {
5076     \cs_if_exist:cTF
5077     {
5078       pgf @ sh @ ns @ \@@_env:
5079       - \int_use:N \l_@@_final_i_int
5080       - \int_eval:n { #2 + #3 }
5081     }
5082     { \int_set:Nn \l_@@_final_j_int { #2 + #3 } }
5083     {
5084       \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5085       \bool_set_true:N \l_@@_final_open_bool
5086     }
5087   }
5088   \group_begin:
5089   \@@_open_shorten:
5090   \int_if_zero:nTF { #1 }
5091   { \color { nicematrix-first-row } }
5092   {
5093     \int_compare:nNnT { #1 } = \g_@@_row_total_int
5094     { \color { nicematrix-last-row } }
5095   }
5096
5097   \keys_set:nn { NiceMatrix / xdots } { #4 }
5098   \tl_if_empty:oF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
5099   \@@_actually_draw_Ldots:
5100   \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).

```

5101 \int_step_inline:nnn { #2 } { #2 + #3 - 1 }
5102 { \cs_set:cpn { @@ _ dotted _ #1 - ##1 } { } }
5103 }

5104 \hook_gput_code:nnn { begindocument } { . }
5105 {
5106   \cs_set_nopar:Npn \l_@@_argspec_tl { m m 0 { } E { _ ^ : } { { } { } { } } }
5107   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
5108   \cs_new_protected:Npn \@@_Vdotsfor:
5109     { \@@_collect_options:n { \@@_Vdotsfor_i } }
5110   \exp_args:NNo \NewDocumentCommand \@@_Vdotsfor_i \l_@@_argspec_tl
5111     {
5112       \bool_gset_true:N \g_@@_empty_cell_bool
5113       \tl_gput_right:Nx \g_@@_HVDotsfor_lines_tl
5114         {
5115           \@@_Vdotsfor:nnnn
5116           { \int_use:N \c@iRow }
5117           { \int_use:N \c@jCol }
5118           { #2 }
5119           {
5120             #1 , #3 ,
5121             down = \exp_not:n { #4 } ,
5122             up = \exp_not:n { #5 } ,
5123             middle = \exp_not:n { #6 }
5124           }
5125         }
5126     }
5127 }

```

```

5128 \cs_new_protected:Npn \@@_Vdotsfor:nnnn #1 #2 #3 #4
5129 {
5130   \bool_set_false:N \l_@@_initial_open_bool
5131   \bool_set_false:N \l_@@_final_open_bool

```

For the column, it's easy.

```

5132 \int_set:Nn \l_@@_initial_j_int { #2 }
5133 \int_set_eq:NN \l_@@_final_j_int \l_@@_initial_j_int

```

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

```

5134 \int_compare:nNnTF { #1 } = \c_one_int
5135 {
5136   \int_set_eq:NN \l_@@_initial_i_int \c_one_int
5137   \bool_set_true:N \l_@@_initial_open_bool
5138 }
5139 {
5140   \cs_if_exist:cTF
5141     {
5142       pgf @ sh @ ns @ \@@_env:
5143       - \int_eval:n { #1 - 1 }
5144       - \int_use:N \l_@@_initial_j_int
5145     }
5146     { \int_set:Nn \l_@@_initial_i_int { #1 - 1 } }
5147     {
5148       \int_set:Nn \l_@@_initial_i_int { #1 }
5149       \bool_set_true:N \l_@@_initial_open_bool
5150     }
5151 }
5152 \int_compare:nNnTF { #1 + #3 - 1 } = \c@iRow
5153 {
5154   \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5155   \bool_set_true:N \l_@@_final_open_bool
5156 }
5157 {

```

```

5158     \cs_if_exist:cTF
5159     {
5160         pgf @ sh @ ns @ \@@_env:
5161         - \int_eval:n { #1 + #3 }
5162         - \int_use:N \l_@@_final_j_int
5163     }
5164     { \int_set:Nn \l_@@_final_i_int { #1 + #3 } }
5165     {
5166         \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5167         \bool_set_true:N \l_@@_final_open_bool
5168     }
5169 }

5170 \group_begin:
5171 \@@_open_shorten:
5172 \int_if_zero:nTF { #2 }
5173 { \color { nicematrix-first-col } }
5174 {
5175     \int_compare:nNnT { #2 } = \g_@@_col_total_int
5176     { \color { nicematrix-last-col } }
5177 }
5178 \keys_set:nn { NiceMatrix / xdots } { #4 }
5179 \tl_if_empty:oF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
5180 \@@_actually_draw_Vdots:
5181 \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).

```

5182     \int_step_inline:nnn { #1 } { #1 + #3 - 1 }
5183     { \cs_set:cpn { @@ _ dotted _ ##1 - #2 } { } }
5184 }

```

The command `\@@_rotate:` will be linked to `\rotate` in `{NiceArrayWithDelims}`.

```

5185 \NewDocumentCommand \@@_rotate: { 0 { } }
5186 {
5187     \peek_remove_spaces:n
5188     {
5189         \bool_gset_true:N \g_@@_rotate_bool
5190         \keys_set:nn { NiceMatrix / rotate } { #1 }
5191     }
5192 }

5193 \keys_define:nn { NiceMatrix / rotate }
5194 {
5195     c .code:n = \bool_gset_true:N \g_@@_rotate_c_bool ,
5196     c .value_forbidden:n = true ,
5197     unknown .code:n = \@@_error:n { Unknown~key~for~rotate }
5198 }

```

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

```

5199 \cs_new:Npn \@@_double_int_eval:n #1-#2 \q_stop
5200 {
5201   \tl_if_empty:nTF { #2 }
5202   { #1 }
5203   { \@@_double_int_eval_i:n #1-#2 \q_stop }
5204 }
5205 \cs_new:Npn \@@_double_int_eval_i:n #1-#2- \q_stop
5206 { \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).

```

5207 \hook_gput_code:nnn { begindocument } { . }
5208 {
5209   \cs_set_nopar:Npn \l_@@_argspec_tl
5210   { 0 { } m m ! 0 { } E { _ ^ : } { { } { } { } } }
5211   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
5212   \exp_args:NNo \NewDocumentCommand \@@_line \l_@@_argspec_tl
5213   {
5214     \group_begin:
5215     \keys_set:nn { NiceMatrix / xdots } { #1 , #4 , down = #5 , up = #6 }
5216     \tl_if_empty:oF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
5217     \use:e
5218     {
5219       \@@_line_i:nn
5220       { \@@_double_int_eval:n #2 - \q_stop }
5221       { \@@_double_int_eval:n #3 - \q_stop }
5222     }
5223     \group_end:
5224   }
5225 }
5226 \cs_new_protected:Npn \@@_line_i:nn #1 #2
5227 {
5228   \bool_set_false:N \l_@@_initial_open_bool
5229   \bool_set_false:N \l_@@_final_open_bool
5230   \bool_lazy_or:nnTF
5231   { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #1 } }
5232   { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #2 } }
5233   { \@@_error:nnn { unknown-cell-for-line-in-CodeAfter } { #1 } { #2 } }
```

The test of `measuring@` is a security (cf. question 686649 on TeX StackExchange).

```

5234   { \legacy_if:nF { measuring@ } { \@@_draw_line_ii:nn { #1 } { #2 } } }
5235 }
5236 \hook_gput_code:nnn { begindocument } { . }
5237 {
5238   \cs_new_protected:Npx \@@_draw_line_ii:nn #1 #2
5239   {
```

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:`.

```

5240   \c_@@_pgfortikzpicture_tl
5241   \@@_draw_line_iii:nn { #1 } { #2 }
```

¹³Indeed, we want that the user may use the command `\line` in `\CodeAfter` with LaTeX counters in the arguments — with the command `\value`.

```

5242         \c_@@_endpgfortikzpicture_tl
5243     }
5244 }

```

The following command *must* be protected (it's used in the construction of `\@@_draw_line_ii:nn`).

```

5245 \cs_new_protected:Npn \@@_draw_line_iii:nn #1 #2
5246 {
5247     \pgfrememberpicturepositiononpagetrue
5248     \pgfpointshapeborder { \@@_env: - #1 } { \@@_qpoint:n { #2 } }
5249     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
5250     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
5251     \pgfpointshapeborder { \@@_env: - #2 } { \@@_qpoint:n { #1 } }
5252     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
5253     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
5254     \@@_draw_line:
5255 }

```

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`

```

5256 \cs_new:Npn \@@_if_row_less_than:nn #1 #2
5257 { \int_compare:nNnT { \c@iRow } < { #1 } { #2 } }

```

`\@@_put_in_row_style` will be used several times by `\RowStyle`.

```

5258 \cs_set_protected:Npn \@@_put_in_row_style:n #1
5259 {
5260     \tl_gput_right:Nx \g_@@_row_style_tl
5261     {

```

Be careful, `\exp_not:N \@@_if_row_less_than:nn` can't be replaced by a protected version of `\@@_if_row_less_than:nn`.

```

5262         \exp_not:N
5263         \@@_if_row_less_than:nn
5264         { \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`).

```

5265         { \exp_not:n { #1 } \scan_stop: }
5266     }
5267 }
5268 \cs_generate_variant:Nn \@@_put_in_row_style:n { e }

```

```

5269 \keys_define:nn { NiceMatrix / RowStyle }
5270 {
5271   cell-space-top-limit .dim_set:N = \l_tmpa_dim ,
5272   cell-space-top-limit .value_required:n = true ,
5273   cell-space-bottom-limit .dim_set:N = \l_tmpb_dim ,
5274   cell-space-bottom-limit .value_required:n = true ,
5275   cell-space-limits .meta:n =
5276   {
5277     cell-space-top-limit = #1 ,
5278     cell-space-bottom-limit = #1 ,
5279   } ,
5280   color .tl_set:N = \l_@@_color_tl ,
5281   color .value_required:n = true ,
5282   bold .bool_set:N = \l_@@_bold_row_style_bool ,
5283   bold .default:n = true ,
5284   nb-rows .code:n =
5285     \str_if_eq:nnTF { #1 } { * }
5286     { \int_set:Nn \l_@@_key_nb_rows_int { 500 } }
5287     { \int_set:Nn \l_@@_key_nb_rows_int { #1 } } ,
5288   nb-rows .value_required:n = true ,
5289   rowcolor .tl_set:N = \l_tmpa_tl ,
5290   rowcolor .value_required:n = true ,
5291   unknown .code:n = \@@_error:n { Unknown~key~for~RowStyle }
5292 }

```

```

5293 \NewDocumentCommand \@@_RowStyle:n { 0 { } m }
5294 {
5295   \group_begin:
5296   \tl_clear:N \l_tmpa_tl
5297   \tl_clear:N \l_@@_color_tl
5298   \int_set_eq:NN \l_@@_key_nb_rows_int \c_one_int
5299   \dim_zero:N \l_tmpa_dim
5300   \dim_zero:N \l_tmpb_dim
5301   \keys_set:nn { NiceMatrix / RowStyle } { #1 }

```

If the key `rowcolor` has been used.

```

5302   \tl_if_empty:NF \l_tmpa_tl
5303   {

```

First, the end of the current row (we remind that `\RowStyle` applies to the *end* of the current row).

```

5304     \tl_gput_right:Nx \g_@@_pre_code_before_tl
5305     {

```

The command `\@@_exp_color_arg:No` is *fully expandable*.

```

5306         \@@_exp_color_arg:No \@@_rectanglecolor \l_tmpa_tl
5307         { \int_use:N \c@iRow - \int_use:N \c@jCol }
5308         { \int_use:N \c@iRow - * }
5309     }

```

Then, the other rows (if there is several rows).

```

5310     \int_compare:nNnT \l_@@_key_nb_rows_int > \c_one_int
5311     {
5312       \tl_gput_right:Nx \g_@@_pre_code_before_tl
5313       {
5314         \@@_exp_color_arg:No \@@_rowcolor \l_tmpa_tl
5315         {
5316           \int_eval:n { \c@iRow + 1 }
5317           - \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int - 1 }
5318         }
5319       }
5320     }
5321   }
5322   \@@_put_in_row_style:n { \exp_not:n { #2 } }

```

`\l_tmpa_dim` is the value of the key `cell-space-top-limit` of `\RowStyle`.

```

5323   \dim_compare:nNnT \l_tmpa_dim > \c_zero_dim
5324   {
5325     \exp_args:Nx \@@_put_in_row_style:n
5326     {
5327       \tl_gput_right:Nn \exp_not:N \g_@@_cell_after_hook_tl
5328       {

```

It's not possible to change the following code by using `\dim_set_eq:NN` (because of expansion).

```

5329         \dim_set:Nn \l_@@_cell_space_top_limit_dim
5330         { \dim_use:N \l_tmpa_dim }
5331       }
5332     }
5333   }

```

`\l_tmpb_dim` is the value of the key `cell-space-bottom-limit` of `\RowStyle`.

```

5334   \dim_compare:nNnT \l_tmpb_dim > \c_zero_dim
5335   {
5336     \exp_args:Nx \@@_put_in_row_style:n
5337     {
5338       \tl_gput_right:Nn \exp_not:N \g_@@_cell_after_hook_tl
5339       {
5340         \dim_set:Nn \l_@@_cell_space_bottom_limit_dim
5341         { \dim_use:N \l_tmpb_dim }
5342       }
5343     }
5344   }

```

`\l_@@_color_tl` is the value of the key `color` of `\RowStyle`.

```

5345   \tl_if_empty:NF \l_@@_color_tl
5346   {
5347     \@@_put_in_row_style:e
5348     {
5349       \mode_leave_vertical:
5350       \@@_color:n { \l_@@_color_tl }
5351     }
5352   }

```

`\l_@@_bold_row_style_bool` is the value of the key `bold`.

```

5353   \bool_if:NT \l_@@_bold_row_style_bool
5354   {
5355     \@@_put_in_row_style:n
5356     {
5357       \exp_not:n
5358       {
5359         \if_mode_math:
5360           \c_math_toggle_token
5361           \bfseries \boldmath
5362           \c_math_toggle_token
5363         \else:
5364           \bfseries \boldmath
5365         \fi:
5366       }
5367     }
5368   }
5369   \group_end:
5370   \g_@@_row_style_tl
5371   \ignorespaces
5372 }

```

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).

```
5373 \cs_new_protected:Npn \@@_add_to_colors_seq:nn #1 #2
5374 {
```

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.

```
5375 \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`.

```
5376 \str_if_in:nnF { #1 } { !! }
5377 {
5378 \seq_map_indexed_inline:Nn \g_@@_colors_seq
5379 { \tl_if_eq:nnT { #1 } { ##2 } { \int_set:Nn \l_tmpa_int { ##1 } } }
5380 }
5381 \int_if_zero:nTF \l_tmpa_int
```

First, the case where the color is a *new* color (not in the sequence).

```
5382 {
5383 \seq_gput_right:Nn \g_@@_colors_seq { #1 }
5384 \tl_gset:cx { g_@@_color _ \seq_count:N \g_@@_colors_seq _ tl } { #2 }
5385 }
```

Now, the case where the color is *not* a new color (the color is in the sequence at the position `\l_tmpa_int`).

```
5386 { \tl_gput_right:cx { g_@@_color _ \int_use:N \l_tmpa_int _ tl } { #2 } }
5387 }

5388 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e n }
5389 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e e }
```

The following command must be used within a `\pgfpicture`.

```
5390 \cs_new_protected:Npn \@@_clip_with_rounded_corners:
5391 {
5392 \dim_compare:nNnT \l_@@_tab_rounded_corners_dim > \c_zero_dim
5393 {
```

The TeX group is for `\pgfsetcornersarced` (whose scope is the TeX scope).

```

5394     \group_begin:
5395     \pgfsetcornersarced
5396     {
5397         \pgfpoint
5398         { \l_@@_tab_rounded_corners_dim }
5399         { \l_@@_tab_rounded_corners_dim }
5400     }

```

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.

```

5401     \bool_if:NTF \l_@@_hvlines_bool
5402     {
5403         \pgfpathrectanglecorners
5404         {
5405             \pgfpointadd
5406             { \@@_qpoint:n { row-1 } }
5407             { \pgfpoint { 0.5 \arrayrulewidth } { \c_zero_dim } }
5408         }
5409         {
5410             \pgfpointadd
5411             {
5412                 \@@_qpoint:n
5413                 { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
5414             }
5415             { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
5416         }
5417     }
5418     {
5419         \pgfpathrectanglecorners
5420         { \@@_qpoint:n { row-1 } }
5421         {
5422             \pgfpointadd
5423             {
5424                 \@@_qpoint:n
5425                 { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
5426             }
5427             { \pgfpoint \c_zero_dim \arrayrulewidth }
5428         }
5429     }
5430     \pgfusepath { clip }
5431     \group_end:

```

The TeX group was for `\pgfsetcornersarced`.

```

5432     }
5433 }

```

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`).

```

5434 \cs_new_protected:Npn \@@_actually_color:
5435 {
5436     \pgfpicture
5437     \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.

```

5438     \@@_clip_with_rounded_corners:
5439     \seq_map_indexed_inline:Nn \g_@@_colors_seq
5440     {
5441         \int_compare:nNnTF { ##1 } = \c_one_int

```

```

5442     {
5443         \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_nocolor:n
5444         \use:c { g_@@_color _ 1 _tl }
5445         \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_normal:n
5446     }
5447     {
5448         \begin { pgfscope }
5449             \@@_color_opacity ##2
5450             \use:c { g_@@_color _ ##1 _tl }
5451             \tl_gclear:c { g_@@_color _ ##1 _tl }
5452             \pgfusepath { fill }
5453         \end { pgfscope }
5454     }
5455 }
5456 \endpgfpicture
5457 }

```

The following command will extract the potential key `opacity` in its optional argument (between square brackets) and (of course) then apply the command `\color`.

```

5458 \cs_new_protected:Npn \@@_color_opacity
5459 {
5460     \peek_meaning:NTF [
5461         { \@@_color_opacity:w }
5462         { \@@_color_opacity:w [ ] }
5463     }

```

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.

```

5464 \cs_new_protected:Npn \@@_color_opacity:w [ #1 ]
5465 {
5466     \tl_clear:N \l_tmpa_tl
5467     \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.

```

5468     \tl_if_empty:NF \l_tmpa_tl { \exp_args:No \pgfsetfillopacity \l_tmpa_tl }
5469     \tl_if_empty:NTF \l_tmpb_tl
5470     { \@declaredcolor }
5471     { \use:e { \exp_not:N \@undeclaredcolor [ \l_tmpb_tl ] } }
5472 }

```

The following set of keys is used by the command `\@@_color_opacity:wn`.

```

5473 \keys_define:nn { nicematrix / color-opacity }
5474 {
5475     opacity .tl_set:N          = \l_tmpa_tl ,
5476     opacity .value_required:n = true
5477 }

5478 \cs_new_protected:Npn \@@_cartesian_color:nn #1 #2
5479 {
5480     \cs_set_nopar:Npn \l_@@_rows_tl { #1 }
5481     \cs_set_nopar:Npn \l_@@_cols_tl { #2 }
5482     \@@_cartesian_path:
5483 }

```

Here is an example : `\@@_rowcolor {red!15} {1,3,5-7,10-}`

```

5484 \NewDocumentCommand \@@_rowcolor { 0 { } m m }
5485 {
5486     \tl_if_blank:nF { #2 }
5487     {

```

```

5488 \@@_add_to_colors_seq:en
5489 { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5490 { \@@_cartesian_color:nn { #3 } { - } }
5491 }
5492 }

```

Here an example : `\@@_columncolor:nn {red!15} {1,3,5-7,10-}`

```

5493 \NewDocumentCommand \@@_columncolor { 0 { } m m }
5494 {
5495   \tl_if_blank:nF { #2 }
5496   {
5497     \@@_add_to_colors_seq:en
5498     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5499     { \@@_cartesian_color:nn { - } { #3 } }
5500   }
5501 }

```

Here is an example : `\@@_rectanglecolor{red!15}{2-3}{5-6}`

```

5502 \NewDocumentCommand \@@_rectanglecolor { 0 { } m m m }
5503 {
5504   \tl_if_blank:nF { #2 }
5505   {
5506     \@@_add_to_colors_seq:en
5507     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5508     { \@@_rectanglecolor:nnn { #3 } { #4 } { \c_zero_dim } }
5509   }
5510 }

```

The last argument is the radius of the corners of the rectangle.

```

5511 \NewDocumentCommand \@@_roundedrectanglecolor { 0 { } m m m m }
5512 {
5513   \tl_if_blank:nF { #2 }
5514   {
5515     \@@_add_to_colors_seq:en
5516     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5517     { \@@_rectanglecolor:nnn { #3 } { #4 } { #5 } }
5518   }
5519 }

```

The last argument is the radius of the corners of the rectangle.

```

5520 \cs_new_protected:Npn \@@_rectanglecolor:nnn #1 #2 #3
5521 {
5522   \@@_cut_on_hyphen:w #1 \q_stop
5523   \tl_clear_new:N \l_@@_tmpc_tl
5524   \tl_clear_new:N \l_@@_tmpd_tl
5525   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
5526   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
5527   \@@_cut_on_hyphen:w #2 \q_stop
5528   \tl_set:Nx \l_@@_rows_tl { \l_@@_tmpc_tl - \l_tmpa_tl }
5529   \tl_set:Nx \l_@@_cols_tl { \l_@@_tmpd_tl - \l_tmpb_tl }

```

The command `\@@_cartesian_path:n` takes in two implicit arguments: `\l_@@_cols_tl` and `\l_@@_rows_tl`.

```

5530 \@@_cartesian_path:n { #3 }
5531 }

```

Here is an example : `\@@_cellcolor[rgb]{0.5,0.5,0}{2-3,3-4,4-5,5-6}`

```

5532 \NewDocumentCommand \@@_cellcolor { 0 { } m m }
5533 {
5534   \clist_map_inline:nn { #3 }
5535   { \@@_rectanglecolor [ #1 ] { #2 } { ##1 } { ##1 } }
5536 }

```

```

5537 \NewDocumentCommand \@@_chessboardcolors { 0 { } m m }
5538 {
5539   \int_step_inline:nn \c@iRow
5540   {
5541     \int_step_inline:nn \c@jCol
5542     {
5543       \int_if_even:nTF { #####1 + ##1 }
5544       { \@@_cellcolor [ #1 ] { #2 } }
5545       { \@@_cellcolor [ #1 ] { #3 } }
5546       { ##1 - #####1 }
5547     }
5548   }
5549 }

```

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”.

```

5550 \NewDocumentCommand \@@_arraycolor { 0 { } m }
5551 {
5552   \@@_rectanglecolor [ #1 ] { #2 }
5553   { 1 - 1 }
5554   { \int_use:N \c@iRow - \int_use:N \c@jCol }
5555 }

```

```

5556 \keys_define:nn { NiceMatrix / rowcolors }
5557 {
5558   respect-blocks .bool_set:N = \l_@@_respect_blocks_bool ,
5559   respect-blocks .default:n = true ,
5560   cols .tl_set:N = \l_@@_cols_tl ,
5561   restart .bool_set:N = \l_@@_rowcolors_restart_bool ,
5562   restart .default:n = true ,
5563   unknown .code:n = \@@_error:n { Unknown-key-for-rowcolors }
5564 }

```

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*.

```

5565 \NewDocumentCommand \@@_rowlistcolors { 0 { } m m 0 { } }
5566 {

```

The group is for the options. `\l_@@_colors_seq` will be the list of colors.

```

5567   \group_begin:
5568   \seq_clear_new:N \l_@@_colors_seq
5569   \seq_set_split:Nnn \l_@@_colors_seq { , } { #3 }
5570   \tl_clear_new:N \l_@@_cols_tl
5571   \cs_set_nopar:Npn \l_@@_cols_tl { - }
5572   \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).

```

5573   \int_zero_new:N \l_@@_color_int
5574   \int_set_eq:NN \l_@@_color_int \c_one_int
5575   \bool_if:NT \l_@@_respect_blocks_bool
5576   {

```

We don't want to take into account a block which is completely in the “first column” (number 0) or in the “last column” and that's why we filter the sequence of the blocks (in a the sequence `\l_tmpa_seq`).

```

5577     \seq_set_eq:NN \l_tmpb_seq \g_@@_pos_of_blocks_seq
5578     \seq_set_filter:NNn \l_tmpa_seq \l_tmpb_seq
5579     { \@@_not_in_exterior_p:nnnnn ##1 }
5580   }
5581   \pgfpicture
5582   \pgf@relevantforpicturesizefalse

```

#2 is the list of intervals of rows.

```

5583   \clist_map_inline:nn { #2 }
5584   {
5585     \cs_set_nopar:Npn \l_tmpa_tl { ##1 }
5586     \tl_if_in:NnTF \l_tmpa_tl { - }
5587     { \@@_cut_on_hyphen:w ##1 \q_stop }
5588     { \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.

```

5589     \int_set:Nn \l_tmpa_int \l_tmpa_tl
5590     \int_set:Nn \l_@@_color_int
5591     { \bool_if:NTF \l_@@_rowcolors_restart_bool 1 \l_tmpa_tl }
5592     \int_zero_new:N \l_@@_tmpc_int
5593     \int_set:Nn \l_@@_tmpc_int \l_tmpb_tl
5594     \int_do_until:nNnn \l_tmpa_int > \l_@@_tmpc_int
5595     {

```

We will compute in `\l_tmpb_int` the last row of the “block”.

```

5596       \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).

```

5597       \bool_if:NT \l_@@_respect_blocks_bool
5598       {
5599         \seq_set_filter:NNn \l_tmpb_seq \l_tmpa_seq
5600         { \@@_intersect_our_row_p:nnnnn #####1 }
5601         \seq_map_inline:Nn \l_tmpb_seq { \@@_rowcolors_i:nnnnn #####1 }

```

Now, the last row of the block is computed in `\l_tmpb_int`.

```

5602       }
5603       \tl_set:No \l_@@_rows_tl
5604       { \int_use:N \l_tmpa_int - \int_use:N \l_tmpb_int }

```

`\l_@@_tmpc_tl` will be the color that we will use.

```

5605       \tl_clear_new:N \l_@@_color_tl
5606       \tl_set:Nx \l_@@_color_tl
5607       {
5608         \@@_color_index:n
5609         {
5610           \int_mod:nn
5611           { \l_@@_color_int - 1 }
5612           { \seq_count:N \l_@@_colors_seq }
5613           + 1
5614         }
5615       }
5616       \tl_if_empty:NF \l_@@_color_tl
5617       {
5618         \@@_add_to_colors_seq:ee
5619         { \tl_if_blank:nF { #1 } { [ #1 ] } { \l_@@_color_tl } }
5620         { \@@_cartesian_color:nn { \l_@@_rows_tl } { \l_@@_cols_tl } }
5621       }
5622       \int_incr:N \l_@@_color_int
5623       \int_set:Nn \l_tmpa_int { \l_tmpb_int + 1 }
5624     }
5625   }
5626   \endpgfpicture

```

```

5627 \group_end:
5628 }

```

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.

```

5629 \cs_new:Npn \@@_color_index:n #1
5630 {
5631   \str_if_eq:eeTF { \seq_item:Nn \l_@@_colors_seq { #1 } } { = }
5632   { \@@_color_index:n { #1 - 1 } }
5633   { \seq_item:Nn \l_@@_colors_seq { #1 } }
5634 }

```

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 curryfication.

```

5635 \NewDocumentCommand \@@_rowcolors { 0 { } m m m }
5636 { \@@_rowlistcolors [ #1 ] { #2 } { { #3 } , { #4 } } }

```

The braces around #3 and #4 are mandatory.

```

5637 \cs_new_protected:Npn \@@_rowcolors_i:nnnnn #1 #2 #3 #4 #5
5638 {
5639   \int_compare:nNnT { #3 } > \l_tmpb_int
5640   { \int_set:Nn \l_tmpb_int { #3 } }
5641 }

```

```

5642 \prg_new_conditional:Nnn \@@_not_in_exterior:nnnnn p
5643 {
5644   \int_if_zero:nTF { #4 }
5645   \prg_return_false:
5646   {
5647     \int_compare:nNnTF { #2 } > \c@jCol
5648     \prg_return_false:
5649     \prg_return_true:
5650   }
5651 }

```

The following command return true when the block intersects the row `\l_tmpa_int`.

```

5652 \prg_new_conditional:Nnn \@@_intersect_our_row:nnnnn p
5653 {
5654   \int_compare:nNnTF { #1 } > \l_tmpa_int
5655   \prg_return_false:
5656   {
5657     \int_compare:nNnTF \l_tmpa_int > { #3 }
5658     \prg_return_false:
5659     \prg_return_true:
5660   }
5661 }

```

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`).

```

5662 \cs_new_protected:Npn \@@_cartesian_path_normal:n #1
5663 {
5664   \dim_compare:nNnTF { #1 } = \c_zero_dim
5665   {
5666     \bool_if:NTF

```

```

5667 \@@_nocolor_used_bool
5668 \@@_cartesian_path_normal_ii:
5669 {
5670   \seq_if_empty:NTF \l_@@_corners_cells_seq
5671   { \@@_cartesian_path_normal_i:n { #1 } }
5672   \@@_cartesian_path_normal_ii:
5673 }
5674 }
5675 { \@@_cartesian_path_normal_i:n { #1 } }
5676 }

```

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.

```

5677 \cs_new_protected:Npn \@@_cartesian_path_normal_i:n #1
5678 {
5679   \pgfsetcornersarced { \pgfpoint { #1 } { #1 } }

```

We begin the loop over the columns.

```

5680 \clist_map_inline:Nn \l_@@_cols_tl
5681 {
5682   \cs_set_nopar:Npn \l_tmpa_tl { ##1 }
5683   \tl_if_in:NnTF \l_tmpa_tl { - }
5684   { \@@_cut_on_hyphen:w ##1 \q_stop }
5685   { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
5686   \tl_if_empty:NTF \l_tmpa_tl
5687   { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5688   {
5689     \tl_if_eq:NNT \l_tmpa_tl \c_@@_star_tl
5690     { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5691   }
5692   \tl_if_empty:NTF \l_tmpb_tl
5693   { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
5694   {
5695     \tl_if_eq:NNT \l_tmpb_tl \c_@@_star_tl
5696     { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
5697   }
5698   \int_compare:nNnT \l_tmpb_tl > \g_@@_col_total_int
5699   { \tl_set:No \l_tmpb_tl { \int_use:N \g_@@_col_total_int } }

```

\l_@@_tmpc_tl will contain the number of column.

```

5700 \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
5701 \@@_qpoint:n { col - \l_tmpa_tl }
5702 \int_compare:nNnTF \l_@@_first_col_int = \l_tmpa_tl
5703 { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
5704 { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
5705 \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
5706 \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }

```

We begin the loop over the rows.

```

5707 \clist_map_inline:Nn \l_@@_rows_tl
5708 {
5709   \cs_set_nopar:Npn \l_tmpa_tl { #####1 }
5710   \tl_if_in:NnTF \l_tmpa_tl { - }
5711   { \@@_cut_on_hyphen:w #####1 \q_stop }
5712   { \@@_cut_on_hyphen:w #####1 - #####1 \q_stop }
5713   \tl_if_empty:NTF \l_tmpa_tl
5714   { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5715   {
5716     \tl_if_eq:NNT \l_tmpa_tl \c_@@_star_tl
5717     { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5718   }
5719   \tl_if_empty:NTF \l_tmpb_tl
5720   { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }
5721   {

```

```

5722         \tl_if_eq:NNT \l_tmpb_tl \c_@@_star_tl
5723         { \tl_set:No \l_tmpb_tl { \int_use:N \c_iRow } }
5724     }
5725     \int_compare:nNt \l_tmpb_tl > \g_@@_row_total_int
5726     { \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`.

```

5727     \cs_if_exist:cF
5728     { @@ _ \l_tmpa_tl _ \l_@@_tmpc_tl _ nocolor }
5729     {
5730         \@@_qpoint:n { row - \int_eval:n { \l_tmpb_tl + 1 } }
5731         \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
5732         \@@_qpoint:n { row - \l_tmpa_tl }
5733         \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
5734         \pgfpathrectanglecorners
5735         { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
5736         { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5737     }
5738 }
5739 }
5740 }

```

Now, the case where the cells will be colored cell by cell (it's mandatory for example if the key `corners` is used).

```

5741 \cs_new_protected:Npn \@@_cartesian_path_normal_ii:
5742 {
5743     \@@_expand_clist:NN \l_@@_cols_tl \c_jCol
5744     \@@_expand_clist:NN \l_@@_rows_tl \c_iRow

```

We begin the loop over the columns.

```

5745     \clist_map_inline:Nn \l_@@_cols_tl
5746     {
5747         \@@_qpoint:n { col - ##1 }
5748         \int_compare:nNtF \l_@@_first_col_int = { ##1 }
5749         { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
5750         { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
5751         \@@_qpoint:n { col - \int_eval:n { ##1 + 1 } }
5752         \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }

```

We begin the loop over the rows.

```

5753     \clist_map_inline:Nn \l_@@_rows_tl
5754     {
5755         \seq_if_in:NnF \l_@@_corners_cells_seq
5756         { #####1 - ##1 }
5757         {
5758             \@@_qpoint:n { row - \int_eval:n { #####1 + 1 } }
5759             \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
5760             \@@_qpoint:n { row - #####1 }
5761             \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
5762             \cs_if_exist:cF
5763             { @@ _ #####1 _ ##1 _ nocolor }
5764             {
5765                 \pgfpathrectanglecorners
5766                 { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
5767                 { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5768             }
5769         }
5770     }
5771 }
5772 }

```

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`).

```

5773 \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

```

5774 \cs_new_protected:Npn \@@_cartesian_path_nocolor:n #1
5775 {
5776   \bool_set_true:N \@@_nocolor_used_bool
5777   \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
5778   \@@_expand_clist:NN \l_@@_rows_tl \c@iRow

```

We begin the loop over the columns.

```

5779   \clist_map_inline:Nn \l_@@_rows_tl
5780   {
5781     \clist_map_inline:Nn \l_@@_cols_tl
5782     { \cs_set:cpn { @@ _ ##1 _ #####1 _ nocolor } { } }
5783   }
5784 }

```

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`.

```

5785 \cs_new_protected:Npn \@@_expand_clist:NN #1 #2
5786 {
5787   \clist_set_eq:NN \l_tmpa_clist #1
5788   \clist_clear:N #1
5789   \clist_map_inline:Nn \l_tmpa_clist
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     \bool_lazy_or:nnT
5796     { \tl_if_blank_p:o \l_tmpa_tl }
5797     { \str_if_eq_p:on \l_tmpa_tl { * } }
5798     { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5799     \bool_lazy_or:nnT
5800     { \tl_if_blank_p:o \l_tmpb_tl }
5801     { \str_if_eq_p:on \l_tmpb_tl { * } }
5802     { \tl_set:Nn \l_tmpb_tl { \int_use:N #2 } }
5803     \int_compare:nNnT \l_tmpb_tl > #2
5804     { \tl_set:Nn \l_tmpb_tl { \int_use:N #2 } }
5805     \int_step_inline:nnn \l_tmpa_tl \l_tmpb_tl
5806     { \clist_put_right:Nn #1 { #####1 } }
5807   }
5808 }

```

When the user uses the key `color-inside`, the following command will be linked to `\cellcolor` in the tabular.

```

5809 \NewDocumentCommand \@@_cellcolor_tabular { 0 { } m }
5810 {
5811   \@@_test_color_inside:
5812   \tl_gput_right:Nx \g_@@_pre_code_before_tl
5813   {

```

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`).

```

5814     \@@_cellcolor [ #1 ] { \exp_not:n { #2 } }
5815     { \int_use:N \c@iRow - \int_use:N \c@jCol }
5816   }
5817   \ignorespaces
5818 }

```

When the user uses the key `color-inside`, the following command will be linked to `\rowcolor` in the tabular.

```

5819 \NewDocumentCommand \@@_rowcolor_tabular { 0 { } m }
5820 {
5821   \@@_test_color_inside:
5822   \tl_gput_right:Nx \g_@@_pre_code_before_tl
5823   {
5824     \@@_rectanglecolor [ #1 ] { \exp_not:n { #2 } }
5825     { \int_use:N \c@iRow - \int_use:N \c@jCol }
5826     { \int_use:N \c@iRow - \exp_not:n { \int_use:N \c@jCol } }
5827   }
5828   \ignorespaces
5829 }

```

When the user uses the key `color-inside`, the following command will be linked to `\rowcolors` in the tabular. The last argument (an optional argument between square brackets is taken by curryfication).

```

5830 \NewDocumentCommand { \@@_rowcolors_tabular } { 0 { } m m }
5831 { \@@_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.

```

5832 \NewDocumentCommand { \@@_rowlistcolors_tabular } { 0 { } m 0 { } }
5833 {
5834   \@@_test_color_inside:
5835   \peek_remove_spaces:n
5836   { \@@_rowlistcolors_tabular:nnn { #1 } { #2 } { #3 } }
5837 }

5838 \cs_new_protected:Npn \@@_rowlistcolors_tabular:nnn #1 #2 #3
5839 {

```

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`.

```

5840 \seq_gclear:N \g_tmpa_seq
5841 \seq_map_inline:Nn \g_@@_rowlistcolors_seq
5842 { \@@_rowlistcolors_tabular_i:nnnn #1 }
5843 \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`.

```

5844 \seq_gput_right:Nx \g_@@_rowlistcolors_seq
5845 {
5846   { \int_use:N \c@iRow }
5847   { \exp_not:n { #1 } }
5848   { \exp_not:n { #2 } }
5849   { restart , cols = \int_use:N \c@jCol - , \exp_not:n { #3 } }
5850 }
5851 }

```

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`).

```

5852 \cs_new_protected:Npn \@@_rowlistcolors_tabular_i:nnnn #1 #2 #3 #4
5853 {
5854   \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).

```

5855   { \seq_gput_right:Nn \g_tmpa_seq { { #1 } { #2 } { #3 } { #4 } } }
5856   {
5857     \tl_gput_right:Nx \g_@@_pre_code_before_tl
5858     {
5859       \@@_rowlistcolors
5860       [ \exp_not:n { #2 } ]
5861       { #1 - \int_eval:n { \c@iRow - 1 } }
5862       { \exp_not:n { #3 } }
5863       [ \exp_not:n { #4 } ]
5864     }
5865   }
5866 }

```

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.

```

5867 \cs_new_protected:Npn \@@_clear_rowlistcolors_seq:
5868 {
5869   \seq_map_inline:Nn \g_@@_rowlistcolors_seq
5870   { \@@_rowlistcolors_tabular_ii:nnnn ##1 }
5871   \seq_gclear:N \g_@@_rowlistcolors_seq
5872 }

5873 \cs_new_protected:Npn \@@_rowlistcolors_tabular_ii:nnnn #1 #2 #3 #4
5874 {
5875   \tl_gput_right:Nn \g_@@_pre_code_before_tl
5876   { \@@_rowlistcolors [ #2 ] { #1 } { #3 } [ #4 ] }
5877 }

```

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.

```

5878 \NewDocumentCommand \@@_columncolor_preamble { 0 { } m }
5879 {

```

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).

```

5880   \int_compare:nNnT \c@jCol > \g_@@_col_total_int
5881   {

```

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).

```

5882   \tl_gput_left:Nx \g_@@_pre_code_before_tl
5883   {
5884     \exp_not:N \columncolor [ #1 ]
5885     { \exp_not:n { #2 } } { \int_use:N \c@jCol }
5886   }
5887 }
5888 }

```

```

5889 \hook_gput_code:nnn { begindocument } { . }
5890 {
5891   \IfPackageLoadedTF { colortbl }
5892   {
5893     \cs_set_eq:NN \@@_old_cellcolor \cellcolor
5894     \cs_set_eq:NN \@@_old_rowcolor \rowcolor
5895     \cs_new_protected:Npn \@@_revert_colortbl:
5896     {
5897       \hook_gput_code:nnn { env / tabular / begin } { . }
5898       {
5899         \cs_set_eq:NN \cellcolor \@@_old_cellcolor
5900         \cs_set_eq:NN \rowcolor \@@_old_rowcolor
5901       }
5902     }
5903   }
5904   { \cs_new_protected:Npn \@@_revert_colortbl: { } }
5905 }

```

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`.

```

5906 \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`.

```

5907 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix:n #1
5908 {
5909   \int_if_zero:nTF \l_@@_first_col_int
5910   { \@@_OnlyMainNiceMatrix_i:n { #1 } }
5911   {
5912     \int_if_zero:nTF \c@jCol
5913     {
5914       \int_compare:nNf \c@iRow = { -1 }
5915       { \int_compare:nNf \c@iRow = { \l_@@_last_row_int - 1 } { #1 } }
5916     }
5917     { \@@_OnlyMainNiceMatrix_i:n { #1 } }
5918   }
5919 }

```

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.

```

5920 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix_i:n #1
5921 {
5922   \int_if_zero:nF \c@iRow
5923   {
5924     \int_compare:nNf \c@iRow = \l_@@_last_row_int
5925     {

```

```

5926         \int_compare:nNtT \c@jCol > \c_zero_int
5927         { \bool_if:NF \l_@@_in_last_col_bool { #1 } }
5928     }
5929 }
5930 }

```

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.

```

5931 \keys_define:nn { NiceMatrix / Rules }
5932 {
5933     position .int_set:N = \l_@@_position_int ,
5934     position .value_required:n = true ,
5935     start .int_set:N = \l_@@_start_int ,
5936     end .code:n =
5937         \bool_lazy_or:nnTF
5938         { \tl_if_empty_p:n { #1 } }
5939         { \str_if_eq_p:nn { #1 } { last } }
5940         { \int_set_eq:NN \l_@@_end_int \c@jCol }
5941         { \int_set:Nn \l_@@_end_int { #1 } }
5942 }

```

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.

```

5943 \keys_define:nn { NiceMatrix / RulesBis }
5944 {
5945     multiplicity .int_set:N = \l_@@_multiplicity_int ,
5946     multiplicity .initial:n = 1 ,
5947     dotted .bool_set:N = \l_@@_dotted_bool ,
5948     dotted .initial:n = false ,
5949     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]`.

```

5950     color .code:n =
5951         \@@_set_CT@arc@:n { #1 }
5952     \tl_set:Nn \l_@@_rule_color_tl { #1 } ,
5953     color .value_required:n = true ,
5954     sep-color .code:n = \@@_set_CT@drsc@:n { #1 } ,
5955     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.

```

5956     tikz .code:n =
5957         \IfPackageLoadedTF { tikz }
5958         { \clist_put_right:Nn \l_@@_tikz_rule_tl { #1 } }
5959         { \@@_error:n { tikz~without~tikz } } ,
5960     tikz .value_required:n = true ,
5961     total-width .dim_set:N = \l_@@_rule_width_dim ,

```

```

5962     total-width .value_required:n = true ,
5963     width .meta:n = { total-width = #1 } ,
5964     unknown .code:n = \@@_error:n { Unknow~key~for~RulesBis }
5965 }

```

The vertical rules

The following command will be executed in the internal `\CodeAfter`. The argument `#1` is a list of `key=value` pairs.

```

5966 \cs_new_protected:Npn \@@_vline:n #1
5967 {

```

The group is for the options.

```

5968     \group_begin:
5969     \int_set_eq:NN \l_@@_end_int \c@iRow
5970     \keys_set_known:nN { NiceMatrix / Rules } { #1 } \l_@@_other_keys_tl

```

The following test is for the case where the user does not use all the columns specified in the preamble of the environment (for instance, a preamble of `|c|c|c|` but only two columns used).

```

5971     \int_compare:nNt \l_@@_position_int < { \c@jCol + 2 }
5972     \@@_vline_i:
5973     \group_end:
5974 }

```

```

5975 \cs_new_protected:Npn \@@_vline_i:
5976 {

```

`\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`.

```

5977     \tl_set:N \l_tmpb_tl { \int_use:N \l_@@_position_int }
5978     \int_step_variable:nnN \l_@@_start_int \l_@@_end_int
5979     \l_tmpa_tl
5980     {

```

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.

```

5981         \bool_gset_true:N \g_tmpa_bool
5982         \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
5983         { \@@_test_vline_in_block:nnnnn ##1 }
5984         \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
5985         { \@@_test_vline_in_block:nnnnn ##1 }
5986         \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
5987         { \@@_test_vline_in_stroken_block:nnnn ##1 }
5988         \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_v:
5989         \bool_if:NTF \g_tmpa_bool
5990         {
5991             \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.

```

5992             { \int_set:Nn \l_@@_local_start_int \l_tmpa_tl }
5993         }
5994     {
5995         \int_compare:nNt \l_@@_local_start_int > \c_zero_int
5996         {
5997             \int_set:Nn \l_@@_local_end_int { \l_tmpa_tl - 1 }
5998             \@@_vline_ii:
5999             \int_zero:N \l_@@_local_start_int
6000         }
6001     }
6002 }
6003 \int_compare:nNt \l_@@_local_start_int > \c_zero_int
6004 {

```

```

6005         \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
6006         \@@_vline_ii:
6007     }
6008 }

6009 \cs_new_protected:Npn \@@_test_in_corner_v:
6010 {
6011     \int_compare:nNnTF \l_tmpb_tl = { \int_eval:n { \c@jCol + 1 } }
6012     {
6013         \seq_if_in:NxT
6014         \l_@@_corners_cells_seq
6015         { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
6016         { \bool_set_false:N \g_tmpa_bool }
6017     }
6018     {
6019         \seq_if_in:NxT
6020         \l_@@_corners_cells_seq
6021         { \l_tmpa_tl - \l_tmpb_tl }
6022         {
6023             \int_compare:nNnTF \l_tmpb_tl = \c_one_int
6024             { \bool_set_false:N \g_tmpa_bool }
6025             {
6026                 \seq_if_in:NxT
6027                 \l_@@_corners_cells_seq
6028                 { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
6029                 { \bool_set_false:N \g_tmpa_bool }
6030             }
6031         }
6032     }
6033 }

6034 \cs_new_protected:Npn \@@_vline_ii:
6035 {
6036     \tl_clear:N \l_@@_tikz_rule_tl
6037     \keys_set:nV { NiceMatrix / RulesBis } \l_@@_other_keys_tl
6038     \bool_if:NTF \l_@@_dotted_bool
6039     \@@_vline_iv:
6040     {
6041         \tl_if_empty:NTF \l_@@_tikz_rule_tl
6042         \@@_vline_iii:
6043         \@@_vline_v:
6044     }
6045 }

```

First the case of a standard rule: the user has not used the key dotted nor the key tikz.

```

6046 \cs_new_protected:Npn \@@_vline_iii:
6047 {
6048     \pgfpicture
6049     \pgfrememberpicturepositiononpagetrue
6050     \pgf@relevantforpicturesizefalse
6051     \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6052     \dim_set_eq:NN \l_tmpa_dim \pgf@y
6053     \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6054     \dim_set:Nn \l_tmpb_dim
6055     {
6056         \pgf@x
6057         - 0.5 \l_@@_rule_width_dim
6058         +
6059         ( \arrayrulewidth * \l_@@_multiplicity_int
6060           + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6061     }

```

```

6062 \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6063 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
6064 \bool_lazy_all:nT
6065 {
6066   { \int_compare_p:nNn \l_@@_multiplicity_int > \c_one_int }
6067   { \cs_if_exist_p:N \CT@drsc@ }
6068   { ! \tl_if_blank_p:o \CT@drsc@ }
6069 }
6070 {
6071   \group_begin:
6072   \CT@drsc@
6073   \dim_add:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }
6074   \dim_sub:Nn \l_@@_tmpc_dim { 0.5 \arrayrulewidth }
6075   \dim_set:Nn \l_@@_tmpd_dim
6076   {
6077     \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
6078     * ( \l_@@_multiplicity_int - 1 )
6079   }
6080   \pgfpathrectanglecorners
6081   { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6082   { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
6083   \pgfusepath { fill }
6084   \group_end:
6085 }
6086 \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6087 \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
6088 \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6089 {
6090   \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
6091   \dim_sub:Nn \l_tmpb_dim \doublerulesep
6092   \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6093   \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
6094 }
6095 \CT@arc@
6096 \pgfsetlinewidth { 1.1 \arrayrulewidth }
6097 \pgfsetrectcap
6098 \pgfusepathqstroke
6099 \endpgfpicture
6100 }

```

The following code is for the case of a dotted rule (with our system of rounded dots).

```

6101 \cs_new_protected:Npn \@@_vline_iv:
6102 {
6103   \pgfpicture
6104   \pgfrememberpicturepositiononpagetrue
6105   \pgf@relevantforpicturesizefalse
6106   \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6107   \dim_set:Nn \l_@@_x_initial_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
6108   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
6109   \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6110   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
6111   \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6112   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
6113   \CT@arc@
6114   \@@_draw_line:
6115   \endpgfpicture
6116 }

```

The following code is for the case when the user uses the key `tikz`.

```

6117 \cs_new_protected:Npn \@@_vline_v:
6118 {
6119   \begin {tikzpicture }
6120   % added 2023/09/25

```

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).

```

6121 \CT@arc@
6122 \tl_if_empty:NF \l_@@_rule_color_tl
6123 { \tl_put_right:Nx \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6124 \pgfrememberpicturepositiononpagetrue
6125 \pgf@relevantforpicturesizefalse
6126 \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6127 \dim_set_eq:NN \l_tmpa_dim \pgf@y
6128 \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6129 \dim_set:Nn \l_tmpb_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
6130 \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6131 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
6132 \exp_args:No \tikzset \l_@@_tikz_rule_tl
6133 \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6134 ( \l_tmpb_dim , \l_tmpa_dim ) --
6135 ( \l_tmpb_dim , \l_@@_tmpc_dim ) ;
6136 \end { tikzpicture }
6137 }

```

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).

```

6138 \cs_new_protected:Npn \@@_draw_vlines:
6139 {
6140   \int_step_inline:nnn
6141   { \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool 2 1 }
6142   {
6143     \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool
6144     \c@jCol
6145     { \int_eval:n { \c@jCol + 1 } }
6146   }
6147   {
6148     \tl_if_eq:NNF \l_@@_vlines_clist \c_@@_all_tl
6149     { \clist_if_in:NnT \l_@@_vlines_clist { ##1 } }
6150     { \@@_vline:n { position = ##1 , total-width = \arrayrulewidth } }
6151   }
6152 }

```

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}`.

```

6153 \cs_new_protected:Npn \@@_hline:n #1
6154 {

```

The group is for the options.

```

6155   \group_begin:
6156   \int_zero_new:N \l_@@_end_int
6157   \int_set_eq:NN \l_@@_end_int \c@jCol
6158   \keys_set_known:nnN { NiceMatrix / Rules } { #1 } \l_@@_other_keys_tl
6159   \@@_hline_i:
6160   \group_end:
6161 }
6162 \cs_new_protected:Npn \@@_hline_i:
6163 {
6164   \int_zero_new:N \l_@@_local_start_int
6165   \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.

```

6166     \tl_set:N \l_tmpa_tl { \int_use:N \l_@@_position_int }
6167     \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
6168     \l_tmpb_tl
6169     {

```

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.

```

6170         \bool_gset_true:N \g_tmpa_bool
6171         \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6172         { \@@_test_hline_in_block:nnnnn ##1 }
6173         \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
6174         { \@@_test_hline_in_block:nnnnn ##1 }
6175         \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
6176         { \@@_test_hline_in_stroken_block:nnnn ##1 }
6177         \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_h:
6178         \bool_if:NTF \g_tmpa_bool
6179         {
6180             \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.

```

6181             { \int_set:Nn \l_@@_local_start_int \l_tmpb_tl }
6182         }
6183         {
6184             \int_compare:nNnT \l_@@_local_start_int > \c_zero_int
6185             {
6186                 \int_set:Nn \l_@@_local_end_int { \l_tmpb_tl - 1 }
6187                 \@@_hline_ii:
6188                 \int_zero:N \l_@@_local_start_int
6189             }
6190         }
6191     }
6192     \int_compare:nNnT \l_@@_local_start_int > \c_zero_int
6193     {
6194         \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
6195         \@@_hline_ii:
6196     }
6197 }

```

```

6198 \cs_new_protected:Npn \@@_test_in_corner_h:
6199 {
6200     \int_compare:nNnTF \l_tmpa_tl = { \int_eval:n { \c@iRow + 1 } }
6201     {
6202         \seq_if_in:NxT
6203         \l_@@_corners_cells_seq
6204         { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
6205         { \bool_set_false:N \g_tmpa_bool }
6206     }
6207     {
6208         \seq_if_in:NxT
6209         \l_@@_corners_cells_seq
6210         { \l_tmpa_tl - \l_tmpb_tl }
6211         {
6212             \int_compare:nNnTF \l_tmpa_tl = \c_one_int
6213             { \bool_set_false:N \g_tmpa_bool }
6214             {
6215                 \seq_if_in:NxT
6216                 \l_@@_corners_cells_seq
6217                 { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }

```

```

6218         { \bool_set_false:N \g_tmpa_bool }
6219     }
6220 }
6221 }
6222 }

6223 \cs_new_protected:Npn \@@_hline_ii:
6224 {
6225     \tl_clear:N \l_@@_tikz_rule_tl
6226     \keys_set:nV { NiceMatrix / RulesBis } \l_@@_other_keys_tl
6227     \bool_if:NTF \l_@@_dotted_bool
6228         \@@_hline_iv:
6229     {
6230         \tl_if_empty:NTF \l_@@_tikz_rule_tl
6231             \@@_hline_iii:
6232             \@@_hline_v:
6233     }
6234 }

```

First the case of a standard rule (without the keys dotted and tikz).

```

6235 \cs_new_protected:Npn \@@_hline_iii:
6236 {
6237     \pgfpicture
6238     \pgfrememberpicturepositiononpagetrue
6239     \pgf@relevantforpicturesizefalse
6240     \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6241     \dim_set_eq:NN \l_tmpa_dim \pgf@x
6242     \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6243     \dim_set:Nn \l_tmpb_dim
6244     {
6245         \pgf@y
6246         - 0.5 \l_@@_rule_width_dim
6247         +
6248         ( \arrayrulewidth * \l_@@_multiplicity_int
6249           + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6250     }
6251     \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6252     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
6253     \bool_lazy_all:nT
6254     {
6255         { \int_compare_p:nNn \l_@@_multiplicity_int > \c_one_int }
6256         { \cs_if_exist_p:N \CT@drsc@ }
6257         { ! \tl_if_blank_p:o \CT@drsc@ }
6258     }
6259     {
6260         \group_begin:
6261         \CT@drsc@
6262         \dim_set:Nn \l_@@_tmpd_dim
6263         {
6264             \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
6265             * ( \l_@@_multiplicity_int - 1 )
6266         }
6267         \pgfpathrectanglecorners
6268         { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6269         { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
6270         \pgfusepathqfill
6271         \group_end:
6272     }
6273     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6274     \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
6275     \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6276     {

```

```

6277     \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
6278     \dim_sub:Nn \l_tmpb_dim \doublerulesep
6279     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6280     \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
6281   }
6282   \CT@arc@
6283   \pgfsetlinewidth { 1.1 \arrayrulewidth }
6284   \pgfsetrectcap
6285   \pgfusepathqstroke
6286   \endpgfpicture
6287 }

```

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}

```

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ \hdots & \hdots & \hdots & \hdots \\ 1 & 2 & 3 & 4 \end{bmatrix}$$

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}

```

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ \hdots & \hdots & \hdots & \hdots \\ 1 & 2 & 3 & 4 \end{bmatrix}$$

```

6288 \cs_new_protected:Npn \@@_hline_iv:
6289 {
6290   \pgfpicture
6291   \pgfrememberpicturepositiononpagetrue
6292   \pgf@relevantforpicturesizefalse
6293   \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6294   \dim_set:Nn \l_@@_y_initial_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
6295   \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
6296   \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6297   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
6298   \int_compare:nNnT \l_@@_local_start_int = \c_one_int
6299   {
6300     \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
6301     \bool_if:NF \g_@@_delims_bool
6302     { \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.

```

6303     \tl_if_eq:NnF \g_@@_left_delim_tl (
6304       { \dim_add:Nn \l_@@_x_initial_dim { 0.5 \l_@@_xdots_inter_dim } }
6305     )
6306     \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6307     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
6308     \int_compare:nNnT \l_@@_local_end_int = \c_j_col
6309     {
6310       \dim_add:Nn \l_@@_x_final_dim \l_@@_right_margin_dim
6311       \bool_if:NF \g_@@_delims_bool
6312       { \dim_add:Nn \l_@@_x_final_dim \arraycolsep }
6313       \tl_if_eq:NnF \g_@@_right_delim_tl (
6314         { \dim_gsub:Nn \l_@@_x_final_dim { 0.5 \l_@@_xdots_inter_dim } }
6315       )
6316     }
6317     \CT@arc@
\@@_draw_line:

```

```

6318 \endpgfpicture
6319 }

```

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`).

```

6320 \cs_new_protected:Npn \@@_hline_v:
6321 {
6322   \begin { tikzpicture }
6323   % added 2023/09/25

```

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).

```

6324   \CT@arc@
6325   \tl_if_empty:NF \l_@@_rule_color_tl
6326   { \tl_put_right:Nx \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6327   \pgfrememberpicturepositiononpagetrue
6328   \pgf@relevantforpicturesizefalse
6329   \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6330   \dim_set_eq:NN \l_tmpa_dim \pgf@x
6331   \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6332   \dim_set:Nn \l_tmpb_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
6333   \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6334   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
6335   \exp_args:No \tikzset \l_@@_tikz_rule_tl
6336   \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6337   ( \l_tmpa_dim , \l_tmpb_dim ) --
6338   ( \l_@@_tmpc_dim , \l_tmpb_dim ) ;
6339   \end { tikzpicture }
6340 }

```

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).

```

6341 \cs_new_protected:Npn \@@_draw_hlines:
6342 {
6343   \int_step_inline:nnn
6344   { \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool 2 1 }
6345   {
6346     \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool
6347     \c@iRow
6348     { \int_eval:n { \c@iRow + 1 } }
6349   }
6350   {
6351     \tl_if_eq:NNF \l_@@_hlines_clist \c_@@_all_tl
6352     { \clist_if_in:NnT \l_@@_hlines_clist { ##1 } }
6353     { \@@_hline:n { position = ##1 , total-width = \arrayrulewidth } }
6354   }
6355 }

```

The command `\@@_Hline:` will be linked to `\Hline` in the environments of `nicematrix`.

```

6356 \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.

```

6357 \cs_set:Npn \@@_Hline_i:n #1
6358 {
6359   \peek_remove_spaces:n
6360   {
6361     \peek_meaning:NTF \Hline
6362     { \@@_Hline_ii:nn { #1 + 1 } }
6363     { \@@_Hline_iii:n { #1 } }
6364   }
6365 }

```

```

6366 \cs_set:Npn \@@_Hline_ii:nn #1 #2 { \@@_Hline_i:n { #1 } }
6367 \cs_set:Npn \@@_Hline_iii:n #1
6368 { \@@_collect_options:n { \@@_Hline_iv:nn { #1 } } }
6369 \cs_set:Npn \@@_Hline_iv:nn #1 #2
6370 {
6371   \@@_compute_rule_width:n { multiplicity = #1 , #2 }
6372   \skip_vertical:N \l_@@_rule_width_dim
6373   \tl_gput_right:Nx \g_@@_pre_code_after_tl
6374   {
6375     \@@_hline:n
6376     {
6377       multiplicity = #1 ,
6378       position = \int_eval:n { \c@iRow + 1 } ,
6379       total-width = \dim_use:N \l_@@_rule_width_dim ,
6380       #2
6381     }
6382   }
6383   \egroup
6384 }

```

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`).

```

6385 \cs_new_protected:Npn \@@_custom_line:n #1
6386 {
6387   \str_clear_new:N \l_@@_command_str
6388   \str_clear_new:N \l_@@_ccommand_str
6389   \str_clear_new:N \l_@@_letter_str
6390   \tl_clear_new:N \l_@@_other_keys_tl
6391   \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.

```

6392   \bool_lazy_all:nTF
6393   {
6394     { \str_if_empty_p:N \l_@@_letter_str }
6395     { \str_if_empty_p:N \l_@@_command_str }
6396     { \str_if_empty_p:N \l_@@_ccommand_str }
6397   }
6398   { \@@_error:n { No~letter~and~no~command } }
6399   { \exp_args:No \@@_custom_line_i:n \l_@@_other_keys_tl }
6400 }
6401 \keys_define:nn { NiceMatrix / custom-line }
6402 {
6403   letter .str_set:N = \l_@@_letter_str ,
6404   letter .value_required:n = true ,
6405   command .str_set:N = \l_@@_command_str ,
6406   command .value_required:n = true ,
6407   ccommand .str_set:N = \l_@@_ccommand_str ,
6408   ccommand .value_required:n = true ,
6409 }
6410 \cs_new_protected:Npn \@@_custom_line_i:n #1
6411 {

```

The following flags will be raised when the keys `tikz`, `dotted` and `color` are used (in the `custom-line`).

```

6412 \bool_set_false:N \l_@@_tikz_rule_bool
6413 \bool_set_false:N \l_@@_dotted_rule_bool
6414 \bool_set_false:N \l_@@_color_bool
6415 \keys_set:nn { NiceMatrix / custom-line-bis } { #1 }
6416 \bool_if:NT \l_@@_tikz_rule_bool
6417 {
6418   \IfPackageLoadedTF { tikz }
6419   { }
6420   { \@@_error:n { tikz-in-custom-line-without-tikz } }
6421   \bool_if:NT \l_@@_color_bool
6422   { \@@_error:n { color-in-custom-line-with-tikz } }
6423 }
6424 \bool_if:NT \l_@@_dotted_rule_bool
6425 {
6426   \int_compare:nNt \l_@@_multiplicity_int > \c_one_int
6427   { \@@_error:n { key-multiplicity-with-dotted } }
6428 }
6429 \str_if_empty:NF \l_@@_letter_str
6430 {
6431   \int_compare:nTF { \str_count:N \l_@@_letter_str != 1 }
6432   { \@@_error:n { Several-letters } }
6433   {
6434     \exp_args:NnV \tl_if_in:NnTF
6435     \c_@@_forbidden_letters_str \l_@@_letter_str
6436     { \@@_error:ne { Forbidden-letter } \l_@@_letter_str }
6437     {

```

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.

```

6438 \cs_set:cpn { @@ _ \l_@@_letter_str } ##1
6439 { \@@_v_custom_line:n { #1 } }
6440 }
6441 }
6442 }
6443 \str_if_empty:NF \l_@@_command_str { \@@_h_custom_line:n { #1 } }
6444 \str_if_empty:NF \l_@@_ccommand_str { \@@_c_custom_line:n { #1 } }
6445 }
6446 \tl_const:Nn \c_@@_forbidden_letters_tl { lcrpmbVX|()[]!@<> }
6447 \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.

```

6448 \keys_define:nn { NiceMatrix / custom-line-bis }
6449 {
6450   multiplicity .int_set:N = \l_@@_multiplicity_int ,
6451   multiplicity .initial:n = 1 ,
6452   multiplicity .value_required:n = true ,
6453   color .code:n = \bool_set_true:N \l_@@_color_bool ,
6454   color .value_required:n = true ,
6455   tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6456   tikz .value_required:n = true ,
6457   dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6458   dotted .value_forbidden:n = true ,
6459   total-width .code:n = { } ,
6460   total-width .value_required:n = true ,
6461   width .code:n = { } ,
6462   width .value_required:n = true ,

```

```

6463     sep-color .code:n = { } ,
6464     sep-color .value_required:n = true ,
6465     unknown .code:n = \@@_error:n { Unknown~key~for~custom~line }
6466 }

```

The following keys will indicate whether the keys `dotted`, `tikz` and `color` are used in the use of a `custom-line`.

```

6467 \bool_new:N \l_@@_dotted_rule_bool
6468 \bool_new:N \l_@@_tikz_rule_bool
6469 \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`.

```

6470 \keys_define:nn { NiceMatrix / custom-line-width }
6471 {
6472     multiplicity .int_set:N = \l_@@_multiplicity_int ,
6473     multiplicity .initial:n = 1 ,
6474     multiplicity .value_required:n = true ,
6475     tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6476     total-width .code:n = \dim_set:Nn \l_@@_rule_width_dim { #1 }
6477                     \bool_set_true:N \l_@@_total_width_bool ,
6478     total-width .value_required:n = true ,
6479     width .meta:n = { total-width = #1 } ,
6480     dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6481 }

```

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`).

```

6482 \cs_new_protected:Npn \@@_h_custom_line:n #1
6483 {

```

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`).

```

6484     \cs_set:cpn { nicematrix - \l_@@_command_str } { \Hline [ #1 ] }
6485     \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_command_str
6486 }

```

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`).

```

6487 \cs_new_protected:Npn \@@_c_custom_line:n #1
6488 {

```

Here, we need an expandable command since it begins with an `\noalign`.

```

6489     \exp_args:Nc \NewExpandableDocumentCommand
6490     { nicematrix - \l_@@_ccommand_str }
6491     { 0 { } m }
6492     {
6493         \noalign
6494         {
6495             \@@_compute_rule_width:n { #1 , ##1 }
6496             \skip_vertical:n { \l_@@_rule_width_dim }
6497             \clist_map_inline:nn
6498             { ##2 }
6499             { \@@_c_custom_line_i:nn { #1 , ##1 } { #####1 } }
6500         }
6501     }
6502     \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_ccommand_str
6503 }

```

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*.

```

6504 \cs_new_protected:Npn \@@_c_custom_line_i:nn #1 #2
6505 {
6506   \str_if_in:nnTF { #2 } { - }
6507   { \@@_cut_on_hyphen:w #2 \q_stop }
6508   { \@@_cut_on_hyphen:w #2 - #2 \q_stop }
6509   \tl_gput_right:Nx \g_@@_pre_code_after_tl
6510   {
6511     \@@_hline:n
6512     {
6513       #1 ,
6514       start = \l_tmpa_tl ,
6515       end = \l_tmpb_tl ,
6516       position = \int_eval:n { \c@iRow + 1 } ,
6517       total-width = \dim_use:N \l_@@_rule_width_dim
6518     }
6519   }
6520 }

6521 \cs_new_protected:Npn \@@_compute_rule_width:n #1
6522 {
6523   \bool_set_false:N \l_@@_tikz_rule_bool
6524   \bool_set_false:N \l_@@_total_width_bool
6525   \bool_set_false:N \l_@@_dotted_rule_bool
6526   \keys_set_known:nn { NiceMatrix / custom-line-width } { #1 }
6527   \bool_if:NF \l_@@_total_width_bool
6528   {
6529     \bool_if:NTF \l_@@_dotted_rule_bool
6530     { \dim_set:Nn \l_@@_rule_width_dim { 2 \l_@@_xdots_radius_dim } }
6531     {
6532       \bool_if:NF \l_@@_tikz_rule_bool
6533       {
6534         \dim_set:Nn \l_@@_rule_width_dim
6535         {
6536           \arrayrulewidth * \l_@@_multiplicity_int
6537           + \doublerulesep * ( \l_@@_multiplicity_int - 1 )
6538         }
6539       }
6540     }
6541   }
6542 }

6543 \cs_new_protected:Npn \@@_v_custom_line:n #1
6544 {
6545   \@@_compute_rule_width:n { #1 }

```

In the following line, the `\dim_use:N` is mandatory since we do an expansion.

```

6546   \tl_gput_right:Nx \g_@@_array_preamble_tl
6547   { \exp_not:N ! { \skip_horizontal:n { \dim_use:N \l_@@_rule_width_dim } } }
6548   \tl_gput_right:Nx \g_@@_pre_code_after_tl
6549   {
6550     \@@_vline:n
6551     {
6552       #1 ,
6553       position = \int_eval:n { \c@jCol + 1 } ,
6554       total-width = \dim_use:N \l_@@_rule_width_dim
6555     }
6556   }
6557   \@@_rec_preamble:n
6558 }

6559 \@@_custom_line:n
6560 { 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.

```

6561 \cs_new_protected:Npn \@@_test_hline_in_block:nnnnn #1 #2 #3 #4 #5
6562 {
6563   \int_compare:nNtT \l_tmpa_tl > { #1 }
6564   {
6565     \int_compare:nNtT \l_tmpa_tl < { #3 + 1 }
6566     {
6567       \int_compare:nNtT \l_tmpb_tl > { #2 - 1 }
6568       {
6569         \int_compare:nNtT \l_tmpb_tl < { #4 + 1 }
6570         { \bool_gset_false:N \g_tmpa_bool }
6571       }
6572     }
6573   }
6574 }

```

The same for vertical rules.

```

6575 \cs_new_protected:Npn \@@_test_vline_in_block:nnnnn #1 #2 #3 #4 #5
6576 {
6577   \int_compare:nNtT \l_tmpa_tl > { #1 - 1 }
6578   {
6579     \int_compare:nNtT \l_tmpa_tl < { #3 + 1 }
6580     {
6581       \int_compare:nNtT \l_tmpb_tl > { #2 }
6582       {
6583         \int_compare:nNtT \l_tmpb_tl < { #4 + 1 }
6584         { \bool_gset_false:N \g_tmpa_bool }
6585       }
6586     }
6587   }
6588 }

6589 \cs_new_protected:Npn \@@_test_hline_in_stroken_block:nnnn #1 #2 #3 #4
6590 {
6591   \int_compare:nNtT \l_tmpb_tl > { #2 - 1 }
6592   {
6593     \int_compare:nNtT \l_tmpb_tl < { #4 + 1 }
6594     {
6595       \int_compare:nNtTF \l_tmpa_tl = { #1 }
6596       { \bool_gset_false:N \g_tmpa_bool }
6597       {
6598         \int_compare:nNtT \l_tmpa_tl = { #3 + 1 }
6599         { \bool_gset_false:N \g_tmpa_bool }
6600       }
6601     }
6602   }
6603 }

6604 \cs_new_protected:Npn \@@_test_vline_in_stroken_block:nnnn #1 #2 #3 #4
6605 {
6606   \int_compare:nNtT \l_tmpa_tl > { #1 - 1 }
6607   {
6608     \int_compare:nNtT \l_tmpa_tl < { #3 + 1 }
6609     {
6610       \int_compare:nNtTF \l_tmpb_tl = { #2 }
6611       { \bool_gset_false:N \g_tmpa_bool }
6612       {
6613         \int_compare:nNtT \l_tmpb_tl = { #4 + 1 }
6614         { \bool_gset_false:N \g_tmpa_bool }
6615       }
6616     }
6617   }
6618 }

```

```

6616     }
6617   }
6618 }

```

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.

```

6619 \cs_new_protected:Npn \@@_compute_corners:
6620 {

```

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.

```

6621   \seq_clear_new:N \l_@@_corners_cells_seq
6622   \clist_map_inline:Nn \l_@@_corners_clist
6623   {
6624     \str_case:nnF { ##1 }
6625     {
6626       { NW }
6627       { \@@_compute_a_corner:nnnnnn 1 1 1 \c@iRow \c@jCol }
6628       { NE }
6629       { \@@_compute_a_corner:nnnnnn 1 \c@jCol 1 { -1 } \c@iRow 1 }
6630       { SW }
6631       { \@@_compute_a_corner:nnnnnn \c@iRow 1 { -1 } 1 1 \c@jCol }
6632       { SE }
6633       { \@@_compute_a_corner:nnnnnn \c@iRow \c@jCol { -1 } { -1 } 1 1 }
6634     }
6635     { \@@_error:nn { bad~corner } { ##1 } }
6636   }

```

Even if the user has used the key `corners` the list of cells in the corners may be empty.

```

6637   \seq_if_empty:NF \l_@@_corners_cells_seq
6638   {

```

You write on the `aux` file the list of the cells which are in the (empty) corners because you need that information in the `\CodeBefore` since the commands which color the `rows`, `columns` and `cells` must not color the cells in the corners.

```

6639     \tl_gput_right:Nx \g_@@_aux_tl
6640     {
6641       \seq_set_from_clist:Nn \exp_not:N \l_@@_corners_cells_seq
6642       { \seq_use:Nnnn \l_@@_corners_cells_seq , , , }
6643     }
6644   }
6645 }

```

“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.

```

6646 \cs_new_protected:Npn \@@_compute_a_corner:nnnnnn #1 #2 #3 #4 #5 #6
6647 {

```

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.

```

6648 \bool_set_false:N \l_tmpa_bool
6649 \int_zero_new:N \l_@@_last_empty_row_int
6650 \int_set:Nn \l_@@_last_empty_row_int { #1 }
6651 \int_step_inline:nnnn { #1 } { #3 } { #5 }
6652 {
6653   \@@_test_if_cell_in_a_block:nn { ##1 } { \int_eval:n { #2 } }
6654   \bool_lazy_or:nnTF
6655   {
6656     \cs_if_exist_p:c
6657     { pgf @ sh @ ns @ \@@_env: - ##1 - \int_eval:n { #2 } }
6658   }
6659   \l_tmpb_bool
6660   { \bool_set_true:N \l_tmpa_bool }
6661   {
6662     \bool_if:NF \l_tmpa_bool
6663     { \int_set:Nn \l_@@_last_empty_row_int { ##1 } }
6664   }
6665 }

```

Now, you determine the last empty cell in the row of number 1.

```

6666 \bool_set_false:N \l_tmpa_bool
6667 \int_zero_new:N \l_@@_last_empty_column_int
6668 \int_set:Nn \l_@@_last_empty_column_int { #2 }
6669 \int_step_inline:nnnn { #2 } { #4 } { #6 }
6670 {
6671   \@@_test_if_cell_in_a_block:nn { \int_eval:n { #1 } } { ##1 }
6672   \bool_lazy_or:nnTF
6673   \l_tmpb_bool
6674   {
6675     \cs_if_exist_p:c
6676     { pgf @ sh @ ns @ \@@_env: - \int_eval:n { #1 } - ##1 }
6677   }
6678   { \bool_set_true:N \l_tmpa_bool }
6679   {
6680     \bool_if:NF \l_tmpa_bool
6681     { \int_set:Nn \l_@@_last_empty_column_int { ##1 } }
6682   }
6683 }

```

Now, we loop over the rows.

```

6684 \int_step_inline:nnnn { #1 } { #3 } \l_@@_last_empty_row_int
6685 {

```

We treat the row number `##1` with another loop.

```

6686 \bool_set_false:N \l_tmpa_bool
6687 \int_step_inline:nnnn { #2 } { #4 } \l_@@_last_empty_column_int
6688 {
6689   \@@_test_if_cell_in_a_block:nn { ##1 } { ####1 }
6690   \bool_lazy_or:nnTF
6691   \l_tmpb_bool
6692   {
6693     \cs_if_exist_p:c
6694     { pgf @ sh @ ns @ \@@_env: - ##1 - ####1 }
6695   }
6696   { \bool_set_true:N \l_tmpa_bool }
6697   {
6698     \bool_if:NF \l_tmpa_bool
6699     {
6700       \int_set:Nn \l_@@_last_empty_column_int { ####1 }

```

```

6701         \seq_put_right:Nn
6702         \l_@@_corners_cells_seq
6703         { ##1 - #####1 }
6704     }
6705 }
6706 }
6707 }
6708 }

```

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`).

```

6709 \cs_new_protected:Npn \@@_test_if_cell_in_a_block:nn #1 #2
6710 {
6711     \int_set:Nn \l_tmpa_int { #1 }
6712     \int_set:Nn \l_tmpb_int { #2 }
6713     \bool_set_false:N \l_tmpb_bool
6714     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6715     { \@@_test_if_cell_in_block:nnnnnnn \l_tmpa_int \l_tmpb_int ##1 }
6716 }
6717 \cs_set_protected:Npn \@@_test_if_cell_in_block:nnnnnnn #1 #2 #3 #4 #5 #6 #7
6718 {
6719     \int_compare:nNnF { #3 } > { #1 }
6720     {
6721         \int_compare:nNnF { #1 } > { #5 }
6722         {
6723             \int_compare:nNnF { #4 } > { #2 }
6724             {
6725                 \int_compare:nNnF { #2 } > { #6 }
6726                 { \bool_set_true:N \l_tmpb_bool }
6727             }
6728         }
6729     }
6730 }

```

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.

```

6731 \bool_new:N \l_@@_block_auto_columns_width_bool

```

Up to now, there is only one option available for the environment `{NiceMatrixBlock}`.

```

6732 \keys_define:nn { NiceMatrix / NiceMatrixBlock }
6733 {
6734     auto-columns-width .code:n =
6735     {
6736         \bool_set_true:N \l_@@_block_auto_columns_width_bool
6737         \dim_gzero_new:N \g_@@_max_cell_width_dim
6738         \bool_set_true:N \l_@@_auto_columns_width_bool
6739     }
6740 }
6741 \NewDocumentEnvironment { NiceMatrixBlock } { ! 0 { } }
6742 {
6743     \int_gincr:N \g_@@_NiceMatrixBlock_int
6744     \dim_zero:N \l_@@_columns_width_dim

```

```

6745 \keys_set:nn { NiceMatrix / NiceMatrixBlock } { #1 }
6746 \bool_if:NT \l_@@_block_auto_columns_width_bool
6747 {
6748   \cs_if_exist:cT
6749   { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
6750   {
6751     % is \exp_args:Nne mandatory?
6752     \exp_args:Nne \dim_set:Nn \l_@@_columns_width_dim
6753     {
6754       \use:c
6755       { @@_max_cell_width _ \int_use:N \g_@@_NiceMatrixBlock_int }
6756     }
6757   }
6758 }
6759 }

```

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).

```

6760 {
6761   \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.

```

6762   { \int_gdecr:N \g_@@_NiceMatrixBlock_int }
6763   {
6764     \bool_if:NT \l_@@_block_auto_columns_width_bool
6765     {
6766       \iow_shipout:Nn \@mainaux \ExplSyntaxOn
6767       \iow_shipout:Nx \@mainaux
6768       {
6769         \cs_gset:cpn
6770         { @@ _ 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.

```

6771         { \dim_eval:n { \g_@@_max_cell_width_dim + \arrayrulewidth } }
6772       }
6773       \iow_shipout:Nn \@mainaux \ExplSyntaxOff
6774     }
6775   }
6776   \ignorespacesafterend
6777 }

```

26 The extra nodes

First, two variants of the functions \dim_min:nn and \dim_max:nn.

```

6778 \cs_generate_variant:Nn \dim_min:nn { v n }
6779 \cs_generate_variant:Nn \dim_max:nn { v n }

```

The following command is called in \@@_use_arraybox_with_notes_c: just before the construction of the blocks (if the creation of medium nodes is required, medium nodes are also created for the blocks and that construction uses the standard medium nodes).

```

6780 \cs_new_protected:Npn \@@_create_extra_nodes:
6781 {
6782   \bool_if:nTF \l_@@_medium_nodes_bool
6783   {
6784     \bool_if:NTF \l_@@_large_nodes_bool
6785     \@@_create_medium_and_large_nodes:
6786     \@@_create_medium_nodes:
6787   }

```

```

6788     { \bool_if:NT \l_@@_large_nodes_bool \@@_create_large_nodes: }
6789 }

```

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`.

```

6790 \cs_new_protected:Npn \@@_computations_for_medium_nodes:
6791 {
6792   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6793   {
6794     \dim_zero_new:c { l_@@_row\_@@_i: _min_dim }
6795     \dim_set_eq:cN { l_@@_row\_@@_i: _min_dim } \c_max_dim
6796     \dim_zero_new:c { l_@@_row\_@@_i: _max_dim }
6797     \dim_set:cn { l_@@_row\_@@_i: _max_dim } { - \c_max_dim }
6798   }
6799   \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
6800   {
6801     \dim_zero_new:c { l_@@_column\_@@_j: _min_dim }
6802     \dim_set_eq:cN { l_@@_column\_@@_j: _min_dim } \c_max_dim
6803     \dim_zero_new:c { l_@@_column\_@@_j: _max_dim }
6804     \dim_set:cn { l_@@_column\_@@_j: _max_dim } { - \c_max_dim }
6805   }

```

We begin the two nested loops over the rows and the columns of the array.

```

6806   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6807   {
6808     \int_step_variable:nnNn
6809     \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.

```

6810     {
6811       \cs_if_exist:cT
6812       { 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`.

```

6813     {
6814       \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { south-west }
6815       \dim_set:cn { l_@@_row\_@@_i: _min_dim }
6816       { \dim_min:vn { l_@@_row _ \@@_i: _min_dim } \pgf@y }
6817       \seq_if_in:NxF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
6818       {
6819         \dim_set:cn { l_@@_column _ \@@_j: _min_dim }
6820         { \dim_min:vn { l_@@_column _ \@@_j: _min_dim } \pgf@x }
6821       }

```

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`.

```

6822     \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { north-east }
6823     \dim_set:cn { l_@@_row _ \@@_i: _ max_dim }
6824     { \dim_max:vn { l_@@_row _ \@@_i: _ max_dim } \pgf@y }
6825     \seq_if_in:NxF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
6826     {
6827         \dim_set:cn { l_@@_column _ \@@_j: _ max_dim }
6828         { \dim_max:vn { l_@@_column _ \@@_j: _ max_dim } \pgf@x }
6829     }
6830 }
6831 }
6832 }

```

Now, we have to deal with empty rows or empty columns since we don't have created nodes in such rows and columns.

```

6833 \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6834 {
6835     \dim_compare:nNnT
6836     { \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } } = \c_max_dim
6837     {
6838         \@@_qpoint:n { row - \@@_i: - base }
6839         \dim_set:cn { l_@@_row _ \@@_i: _ max _ dim } \pgf@y
6840         \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim } \pgf@y
6841     }
6842 }
6843 \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
6844 {
6845     \dim_compare:nNnT
6846     { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } } = \c_max_dim
6847     {
6848         \@@_qpoint:n { col - \@@_j: }
6849         \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim } \pgf@y
6850         \dim_set:cn { l_@@_column _ \@@_j: _ min _ dim } \pgf@y
6851     }
6852 }
6853 }

```

Here is the command `\@@_create_medium_nodes:`. When this command is used, the “medium nodes” are created.

```

6854 \cs_new_protected:Npn \@@_create_medium_nodes:
6855 {
6856     \pgfpicture
6857     \pgfrememberpicturepositiononpagetrue
6858     \pgf@relevantforpicturesizefalse
6859     \@@_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”.

```

6860     \cs_set_nopar:Npn \l_@@_suffix_tl { -medium }
6861     \@@_create_nodes:
6862     \endpgfpicture
6863 }

```

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:`.

¹⁴If we want to create both, we have to use `\@@_create_medium_and_large_nodes:`

```

6864 \cs_new_protected:Npn \@@_create_large_nodes:
6865 {
6866   \pgfpicture
6867     \pgfrememberpicturepositiononpagetrue
6868     \pgf@relevantforpicturesizefalse
6869     \@@_computations_for_medium_nodes:
6870     \@@_computations_for_large_nodes:
6871     \cs_set_nopar:Npn \l_@@_suffix_tl { - large }
6872     \@@_create_nodes:
6873   \endpgfpicture
6874 }
6875 \cs_new_protected:Npn \@@_create_medium_and_large_nodes:
6876 {
6877   \pgfpicture
6878     \pgfrememberpicturepositiononpagetrue
6879     \pgf@relevantforpicturesizefalse
6880     \@@_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”.

```

6881     \cs_set_nopar:Npn \l_@@_suffix_tl { - medium }
6882     \@@_create_nodes:
6883     \@@_computations_for_large_nodes:
6884     \cs_set_nopar:Npn \l_@@_suffix_tl { - large }
6885     \@@_create_nodes:
6886   \endpgfpicture
6887 }

```

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.

```

6888 \cs_new_protected:Npn \@@_computations_for_large_nodes:
6889 {
6890   \int_set_eq:NN \l_@@_first_row_int \c_one_int
6891   \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`.

```

6892   \int_step_variable:nNn { \c@iRow - 1 } \@@_i:
6893   {
6894     \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim }
6895     {
6896       (
6897         \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } +
6898         \dim_use:c { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
6899       )
6900       / 2
6901     }
6902     \dim_set_eq:cc { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
6903     { l_@@_row _ \@@_i: _ min _ dim }
6904   }
6905   \int_step_variable:nNn { \c@jCol - 1 } \@@_j:
6906   {
6907     \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim }
6908     {
6909       (
6910         \dim_use:c { l_@@_column _ \@@_j: _ max _ dim } +
6911         \dim_use:c
6912           { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
6913       )
6914       / 2
6915     }
6916     \dim_set_eq:cc { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
6917     { l_@@_column _ \@@_j: _ max _ dim }
6918   }

```

Here, we have to use `\dim_sub:cn` because of the number 1 in the name.

```

6919   \dim_sub:cn
6920   { l_@@_column _ 1 _ min _ dim }
6921   \l_@@_left_margin_dim
6922   \dim_add:cn
6923   { l_@@_column _ \int_use:N \c@jCol _ max _ dim }
6924   \l_@@_right_margin_dim
6925 }

```

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).

```

6926 \cs_new_protected:Npn \@@_create_nodes:
6927 {
6928   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6929   {
6930     \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
6931     {

```

We draw the rectangular node for the cell (`\@@_i-\@@_j`).

```

6932       \@@_pgf_rect_node:nnnnn
6933       { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
6934       { \dim_use:c { l_@@_column_ \@@_j: _min_dim } }
6935       { \dim_use:c { l_@@_row_ \@@_i: _min_dim } }
6936       { \dim_use:c { l_@@_column_ \@@_j: _max_dim } }
6937       { \dim_use:c { l_@@_row_ \@@_i: _max_dim } }
6938       \str_if_empty:NF \l_@@_name_str
6939       {
6940         \pgfnodealias
6941         { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
6942         { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
6943       }
6944     }
6945   }

```

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 .

```

6946   \seq_map_pairwise_function:NNN
6947   \g_@@_multicolumn_cells_seq
6948   \g_@@_multicolumn_sizes_seq
6949   \@@_node_for_multicolumn:nn
6950 }

```

```

6951 \cs_new_protected:Npn \@@_extract_coords_values: #1 - #2 \q_stop
6952 {
6953   \cs_set_nopar:Npn \@@_i: { #1 }
6954   \cs_set_nopar:Npn \@@_j: { #2 }
6955 }

```

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”).

```

6956 \cs_new_protected:Npn \@@_node_for_multicolumn:nn #1 #2
6957 {
6958   \@@_extract_coords_values: #1 \q_stop
6959   \@@_pgf_rect_node:nnnnn
6960   { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
6961   { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } }

```

```

6962     { \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } }
6963     { \dim_use:c { l_@@_column _ \int_eval:n { \@@_j: +#2-1 } _ max _ dim } }
6964     { \dim_use:c { l_@@_row _ \@@_i: _ max _ dim } }
6965 \str_if_empty:NF \l_@@_name_str
6966 {
6967     \pgfnodealias
6968     { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
6969     { \int_use:N \g_@@_env_int - \@@_i: - \@@_j: \l_@@_suffix_tl}
6970 }
6971 }

```

27 The blocks

The 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.

```

6972 \keys_define:nn { NiceMatrix / Block / FirstPass }
6973 {
6974     l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
6975     l .value_forbidden:n = true ,
6976     r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
6977     r .value_forbidden:n = true ,
6978     c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
6979     c .value_forbidden:n = true ,
6980     L .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
6981     L .value_forbidden:n = true ,
6982     R .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
6983     R .value_forbidden:n = true ,
6984     C .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
6985     C .value_forbidden:n = true ,
6986     t .code:n = \str_set:Nn \l_@@_vpos_block_str t ,
6987     t .value_forbidden:n = true ,
6988     T .code:n = \str_set:Nn \l_@@_vpos_block_str T ,
6989     T .value_forbidden:n = true ,
6990     b .code:n = \str_set:Nn \l_@@_vpos_block_str b ,
6991     b .value_forbidden:n = true ,
6992     B .code:n = \str_set:Nn \l_@@_vpos_block_str B ,
6993     B .value_forbidden:n = true ,
6994     color .code:n =
6995         \@@_color:n { #1 }
6996         \tl_set_rescan:Nnn
6997             \l_@@_draw_tl
6998             { \char_set_catcode_other:N ! }
6999             { #1 } ,
7000     color .value_required:n = true ,
7001     respect_arraystretch .code:n =
7002         \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
7003     respect_arraystretch .value_forbidden:n = true ,
7004 }

```

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.

```

7005 \cs_new_protected:Npn \@@_Block: { \@@_collect_options:n { \@@_Block_i: } }

7006 \NewExpandableDocumentCommand \@@_Block_i: { m m D < > { } +m }
7007 {

```

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).

```

7008 \peek_remove_spaces:n
7009 {
7010   \tl_if_blank:nTF { #2 }
7011   { \@@_Block_ii:nnnnn \c_one_int \c_one_int }
7012   {
7013     \int_compare:nNnTF { \char_value_catcode:n { 45 } } = { 13 }
7014     \@@_Block_i_czech \@@_Block_i
7015     #2 \q_stop
7016   }
7017   { #1 } { #3 } { #4 }
7018 }
7019 }

```

With the following construction, we extract the values of i and j in the first mandatory argument of the command.

```

7020 \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`.

```

7021 {
7022   \char_set_catcode_active:N -
7023   \cs_new:Npn \@@_Block_i_czech #1-#2 \q_stop { \@@_Block_ii:nnnnn { #1 } { #2 } }
7024 }

```

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.

```

7025 \cs_new_protected:Npn \@@_Block_ii:nnnnn #1 #2 #3 #4 #5
7026 {

```

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).

```

7027 \bool_lazy_or:nnTF
7028 { \tl_if_blank_p:n { #1 } }
7029 { \str_if_eq_p:nn { #1 } { * } }
7030 { \int_set:Nn \l_tmpa_int { 100 } }
7031 { \int_set:Nn \l_tmpa_int { #1 } }
7032 \bool_lazy_or:nnTF
7033 { \tl_if_blank_p:n { #2 } }
7034 { \str_if_eq_p:nn { #2 } { * } }
7035 { \int_set:Nn \l_tmpb_int { 100 } }
7036 { \int_set:Nn \l_tmpb_int { #2 } }

```

If the block is mono-column.

```

7037 \int_compare:nNnTF \l_tmpb_int = \c_one_int
7038 {
7039   \tl_if_empty:NTF \l_@@_hpos_cell_tl
7040   { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_c_str }
7041   { \str_set:NV \l_@@_hpos_block_str \l_@@_hpos_cell_tl }
7042 }
7043 { \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.

```

7044 \keys_set_known:nn { NiceMatrix / Block / FirstPass } { #3 }

```

```

7045 \tl_set:Nx \l_tmpa_tl
7046 {
7047   { \int_use:N \c@iRow }
7048   { \int_use:N \c@jCol }
7049   { \int_eval:n { \c@iRow + \l_tmpa_int - 1 } }
7050   { \int_eval:n { \c@jCol + \l_tmpb_int - 1 } }
7051 }

```

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}`.

If the block is mono-column or mono-row, we have a special treatment. That’s why we have two macros: `\@@_Block_iv:nnnnn` and `\@@_Block_v:nnnnn` (the five arguments of those macros are provided by curryfication).

```

7052 \bool_if:nTF
7053 {
7054   (
7055     \int_compare_p:nNn \l_tmpa_int = \c_one_int
7056     ||
7057     \int_compare_p:nNn \l_tmpb_int = \c_one_int
7058   )
7059   && ! \tl_if_empty_p:n { #5 }

```

For the blocks mono-column, we will compose right now in a box in order to compute its width and take that width into account for the width of the column. However, if the column is a **X** column, we should not do that since the width is determined by another way. This should be the same for the **p**, **m** and **b** columns and we should modify that point. However, for the **X** column, it’s imperative. Otherwise, the process for the determination of the widths of the columns will be wrong.

```

7060   && ! \l_@@_X_bool
7061 }
7062 { \exp_args:Nee \@@_Block_iv:nnnnn }
7063 { \exp_args:Nee \@@_Block_v:nnnnn }
7064 { \l_tmpa_int } { \l_tmpb_int } { #3 } { #4 } { #5 }
7065 }

```

The following macro is for the case of a `\Block` which is mono-row or mono-column (or both). 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.

```

7066 \cs_new_protected:Npn \@@_Block_iv:nnnnn #1 #2 #3 #4 #5
7067 {
7068   \int_gincr:N \g_@@_block_box_int
7069   \cs_set_protected_nopar:Npn \diagbox ##1 ##2
7070   {
7071     \tl_gput_right:Nx \g_@@_pre_code_after_tl
7072     {
7073       \@@_actually_diagbox:nnnnnn
7074       { \int_use:N \c@iRow }
7075       { \int_use:N \c@jCol }
7076       { \int_eval:n { \c@iRow + #1 - 1 } }
7077       { \int_eval:n { \c@jCol + #2 - 1 } }
7078       { \g_@@_row_style_tl \exp_not:n { ##1 } }
7079       { \g_@@_row_style_tl \exp_not:n { ##2 } }
7080     }
7081   }
7082   \box_gclear_new:c
7083   { \g_@@_block_box_int \int_use:N \g_@@_block_box_int _box }

```

Now, we will actually compose the content of the `\Block` in a TeX box. *Be careful*: if after the construction of the box, the boolean `\g_@@_rotate_bool` is raised (which means that the command `\rotate` was present in the content of the `\Block`) we will rotate the box but also, maybe, change the position of the baseline!

```

7084 \hbox_gset:cn
7085 { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
7086 {

```

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}`).

```

7087 \tl_if_empty:NTF \l_@@_color_tl
7088 { \int_compare:nNnT { #2 } = \c_one_int \set@color }
7089 { \@@_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`.

```

7090 \int_compare:nNnT { #1 } = \c_one_int
7091 {
7092   \int_if_zero:nTF \c@iRow
7093   \l_@@_code_for_first_row_tl
7094   {
7095     \int_compare:nNnT \c@iRow = \l_@@_last_row_int
7096     \l_@@_code_for_last_row_tl
7097   }
7098   \g_@@_row_style_tl
7099 }

```

The following command will be no-op when `respect-arraystretch` is in force.

```

7100 \@@_reset_arraystretch:
7101 \dim_zero:N \extrarowheight

```

#4 is the optional argument of the command `\Block`, provided with the syntax `<...>`.

```

7102 #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.).

```

7103 \@@_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}`.

```

7104 \bool_if:NTF \l_@@_tabular_bool
7105 {
7106   \bool_lazy_all:nTF
7107   {
7108     { \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`.

```

7109 { ! \dim_compare_p:nNn \l_@@_col_width_dim < \c_zero_dim }
7110 { ! \g_@@_rotate_bool }
7111 }

```

When the block is mono-column in a column with a fixed width (eg `p{3cm}`), we use a `{minipage}`.

```

7112 {
7113   \use:e
7114   {
7115     \exp_not:N \begin { minipage }%
7116     [ \str_lowercase:V \l_@@_vpos_block_str ]
7117     { \l_@@_col_width_dim }

```

```

7118         \str_case:on \l_@@_hpos_block_str
7119         { c \centering r \raggedleft l \raggedright }
7120     }
7121     #5
7122     \end { minipage }
7123 }

```

In the other cases, we use a `{tabular}`.

```

7124     {
7125         \use:e
7126         {
7127             \exp_not:N \begin { tabular }%
7128             [ \str_lowercase:V \l_@@_vpos_block_str ]
7129             { @ { } \l_@@_hpos_block_str @ { } }
7130         }
7131         #5
7132         \end { tabular }
7133     }
7134 }

```

If we are in a mathematical array (`\l_@@_tabular_bool` is false). The composition is always done with an `{array}` (never with a `{minipage}`).

```

7135     {
7136         \c_math_toggle_token
7137         \use:e
7138         {
7139             \exp_not:N \begin { array }%
7140             [ \str_lowercase:V \l_@@_vpos_block_str ]
7141             { @ { } \l_@@_hpos_block_str @ { } }
7142         }
7143         #5
7144         \end { array }
7145         \c_math_toggle_token
7146     }
7147 }

```

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 the rotated box).

```

7148     \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.

```

7149     \int_compare:nNnT { #2 } = \c_one_int
7150     {
7151         \dim_gset:Nn \g_@@_blocks_wd_dim
7152         {
7153             \dim_max:nn
7154             \g_@@_blocks_wd_dim
7155             {
7156                 \box_wd:c
7157                 { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7158             }
7159         }
7160     }

```

If we are in a mono-row block and if that block has no vertical option for the position¹⁵, we take into account the height and the depth of that block for the height and the depth of the row.

```

7161     \str_if_eq:VnT \l_@@_vpos_block_str { c }

```

¹⁵If the block has a key of a vertical position, that means that it has to be put in a vertical space determined by the *others* cells of the row. Therefore there is no point creating space here. Moreover, that would lead to problems when a multi-row block with a position key such as `b` or `B`.

```

7162 {
7163     \int_compare:nNnT { #1 } = \c_one_int
7164     {
7165         \dim_gset:Nn \g_@@_blocks_ht_dim
7166         {
7167             \dim_max:nn
7168             \g_@@_blocks_ht_dim
7169             {
7170                 \box_ht:c
7171                 { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7172             }
7173         }
7174         \dim_gset:Nn \g_@@_blocks_dp_dim
7175         {
7176             \dim_max:nn
7177             \g_@@_blocks_dp_dim
7178             {
7179                 \box_dp:c
7180                 { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7181             }
7182         }
7183     }
7184 }
7185 \seq_gput_right:Nx \g_@@_blocks_seq
7186 {
7187     \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.

```

7188 {
7189     \exp_not:n { #3 } ,
7190     \l_@@_hpos_block_str ,

```

Now, we put a key for the vertical alignment.

```

7191     \bool_if:NT \g_@@_rotate_bool
7192     {
7193         \bool_if:NTF \g_@@_rotate_c_bool
7194         { v-center }
7195         { \int_compare:nNnT \c@iRow = \l_@@_last_row_int T }
7196     }
7197 }
7198 {
7199     {
7200         \box_use_drop:c
7201         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7202     }
7203 }
7204 \bool_set_false:N \g_@@_rotate_c_bool
7205 }

```

```

7206 \cs_new:Npn \@@_adjust_hpos_rotate:
7207 {
7208     \bool_if:NT \g_@@_rotate_bool
7209     {
7210         \str_set:Nx \l_@@_hpos_block_str
7211         {
7212             \bool_if:NTF \g_@@_rotate_c_bool
7213             { c }
7214             {
7215                 \str_case:onF \l_@@_vpos_block_str
7216                 { b l B l t r T r }

```

```

7217         { \int_compare:nNnTF \c@iRow = \l_@@_last_row_int r l }
7218     }
7219 }
7220 }
7221 }

```

Despite its name the following command rotates the box of the block *but also does vertical adjustment of the baseline of the block*.

```

7222 \cs_new_protected:Npn \@@_rotate_box_of_block:
7223 {
7224     \box_grotate:cn
7225     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7226     { 90 }
7227     \int_compare:nNnT \c@iRow = \l_@@_last_row_int
7228     {
7229         \vbox_gset_top:cn
7230         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7231         {
7232             \skip_vertical:n { 0.8 ex }
7233             \box_use:c
7234             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7235         }
7236     }
7237     \bool_if:NT \g_@@_rotate_c_bool
7238     {
7239         \hbox_gset:cn
7240         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7241         {
7242             \c_math_toggle_token
7243             \vcenter
7244             {
7245                 \box_use:c
7246                 { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7247             }
7248             \c_math_toggle_token
7249         }
7250     }
7251 }

```

The following macro is for the standard case, where the block is not mono-row and not mono-column. In that case, the content of the block is *not* composed right now in a box. The composition in a box will be done further, just after the construction of the array (cf. `\@@_draw_blocks:` and above all `\@@_Block_v:nnnnnn`).

#1 is i (the number of rows of the block), #2 is j (the number of columns of the block), #3 is the list of *key=values* pairs, #4 are the tokens to put before the math mode and before the composition of the block and #5 is the label (=content) of the block.

```

7252 \cs_new_protected:Npn \@@_Block_v:nnnnn #1 #2 #3 #4 #5
7253 {
7254     \seq_gput_right:Nx \g_@@_blocks_seq
7255     {
7256         \l_tmpa_tl
7257         { \exp_not:n { #3 } }
7258         {
7259             \bool_if:NTF \l_@@_tabular_bool
7260             {
7261                 \group_begin:

```

The following command will be no-op when `respect-arraystretch` is in force.

```

7262     \@@_reset_arraystretch:
7263     \exp_not:n
7264     {
7265         \dim_zero:N \extrarowheight
7266         #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).

```

7267         \use:e
7268         {
7269             \exp_not:N \begin { tabular } [ \l_@@_vpos_block_str ]
7270             { @ { } \l_@@_hpos_block_str @ { } }
7271         }
7272         #5
7273         \end { tabular }
7274     }
7275     \group_end:
7276 }

```

When we are *not* in an environments `{NiceTabular}` (or similar).

```

7277     {
7278         \group_begin:

```

The following will be no-op when `respect-arraystretch` is in force.

```

7279         \@@_reset_arraystretch:
7280         \exp_not:n
7281         {
7282             \dim_zero:N \extrarowheight
7283             #4
7284             \c_math_toggle_token
7285             \use:e
7286             {
7287                 \exp_not:N \begin { array } [ \l_@@_vpos_block_str ]
7288                 { @ { } \l_@@_hpos_block_str @ { } }
7289             }
7290             #5
7291             \end { array }
7292             \c_math_toggle_token
7293         }
7294         \group_end:
7295     }
7296 }
7297 }
7298 }

```

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).

```

7299 \keys_define:nn { NiceMatrix / Block / SecondPass }
7300 {
7301     tikz .code:n =
7302         \IfPackageLoadedTF { tikz }
7303         { \seq_put_right:Nn \l_@@_tikz_seq { { #1 } } }
7304         { \@@_error:n { tikz~key~without~tikz } } ,
7305     tikz .value_required:n = true ,
7306     fill .code:n =
7307         \tl_set_rescan:Nnn
7308         \l_@@_fill_tl
7309         { \char_set_catcode_other:N ! }
7310         { #1 } ,
7311     fill .value_required:n = true ,
7312     opacity .tl_set:N = \l_@@_opacity_tl ,
7313     opacity .value_required:n = true ,
7314     draw .code:n =
7315         \tl_set_rescan:Nnn

```

```

7316 \l_@@_draw_tl
7317 { \char_set_catcode_other:N ! }
7318 { #1 } ,
7319 draw .default:n = default ,
7320 rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
7321 rounded-corners .default:n = 4 pt ,
7322 color .code:n =
7323   \@@_color:n { #1 }
7324   \tl_set_rescan:Nnn
7325     \l_@@_draw_tl
7326     { \char_set_catcode_other:N ! }
7327     { #1 } ,
7328 borders .clist_set:N = \l_@@_borders_clist ,
7329 borders .value_required:n = true ,
7330 hvlines .meta:n = { vlines , hlines } ,
7331 vlines .bool_set:N = \l_@@_vlines_block_bool ,
7332 vlines .default:n = true ,
7333 hlines .bool_set:N = \l_@@_hlines_block_bool ,
7334 hlines .default:n = true ,
7335 line-width .dim_set:N = \l_@@_line_width_dim ,
7336 line-width .value_required:n = true ,

```

Some keys have not a property `.value_required:n` (or similar) because they are in `FirstPass`.

```

7337 l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7338 r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7339 c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7340 L .code:n = \str_set:Nn \l_@@_hpos_block_str l
7341   \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7342 R .code:n = \str_set:Nn \l_@@_hpos_block_str r
7343   \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7344 C .code:n = \str_set:Nn \l_@@_hpos_block_str c
7345   \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7346 t .code:n = \str_set:Nn \l_@@_vpos_block_str t ,
7347 T .code:n = \str_set:Nn \l_@@_vpos_block_str T ,
7348 b .code:n = \str_set:Nn \l_@@_vpos_block_str b ,
7349 B .code:n = \str_set:Nn \l_@@_vpos_block_str B ,
7350 v-center .code:n = \str_set:Nn \l_@@_vpos_block_str { c } ,
7351 v-center .value_forbidden:n = true ,
7352 name .tl_set:N = \l_@@_block_name_str ,
7353 name .value_required:n = true ,
7354 name .initial:n = ,
7355 respect-arraystretch .code:n =
7356   \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
7357 respect-arraystretch .value_forbidden:n = true ,
7358 transparent .bool_set:N = \l_@@_transparent_bool ,
7359 transparent .default:n = true ,
7360 transparent .initial:n = false ,
7361 unknown .code:n = \@@_error:n { Unknown~key~for~Block }
7362 }

```

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.

```

7363 \cs_new_protected:Npn \@@_draw_blocks:
7364 {
7365   \cs_set_eq:NN \ialign \@@_old_ialign:
7366   \seq_map_inline:Nn \g_@@_blocks_seq { \@@_Block_iv:nnnnnn ##1 }
7367 }
7368 \cs_new_protected:Npn \@@_Block_iv:nnnnnn #1 #2 #3 #4 #5 #6
7369 {

```

The integer `\l_@@_last_row_int` will be the last row of the block and `\l_@@_last_col_int` its last column.

```

7370 \int_zero_new:N \l_@@_last_row_int
7371 \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.

```

7372 \int_compare:nNnTF { #3 } > { 99 }
7373 { \int_set_eq:NN \l_@@_last_row_int \c@iRow }
7374 { \int_set:Nn \l_@@_last_row_int { #3 } }
7375 \int_compare:nNnTF { #4 } > { 99 }
7376 { \int_set_eq:NN \l_@@_last_col_int \c@jCol }
7377 { \int_set:Nn \l_@@_last_col_int { #4 } }
7378 \int_compare:nNnTF \l_@@_last_col_int > \g_@@_col_total_int
7379 {
7380   \bool_lazy_and:nnTF
7381     \l_@@_preamble_bool
7382     {
7383       \int_compare_p:n
7384         { \l_@@_last_col_int <= \g_@@_static_num_of_col_int }
7385     }
7386     {
7387       \msg_error:nnnn { nicematrix } { Block-too-large-2 } { #1 } { #2 }
7388       \@@_msg_redirect_name:nn { Block-too-large-2 } { none }
7389       \@@_msg_redirect_name:nn { columns-not-used } { none }
7390     }
7391   { \msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 } }
7392 }
7393 {
7394   \int_compare:nNnTF \l_@@_last_row_int > \g_@@_row_total_int
7395   { \msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 } }
7396   { \@@_Block_v:nnnnnn { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } }
7397 }
7398 }

```

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

```

7399 \cs_new_protected:Npn \@@_Block_v:nnnnnn #1 #2 #3 #4 #5 #6
7400 {

```

The group is for the keys.

```

7401   \group_begin:
7402   \int_compare:nNnT { #1 } = { #3 }
7403   { \str_set:Nn \l_@@_vpos_block_str { t } }
7404   \keys_set:nn { NiceMatrix / Block / SecondPass } { #5 }
7405   \bool_if:NT \l_@@_vlines_block_bool
7406   {
7407     \tl_gput_right:Nx \g_nicematrix_code_after_tl
7408     {
7409       \@@_vlines_block:nnn
7410       { \exp_not:n { #5 } }
7411       { #1 - #2 }
7412       { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7413     }
7414   }
7415   \bool_if:NT \l_@@_hlines_block_bool
7416   {
7417     \tl_gput_right:Nx \g_nicematrix_code_after_tl
7418     {
7419       \@@_hlines_block:nnn

```

```

7420         { \exp_not:n { #5 } }
7421         { #1 - #2 }
7422         { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7423     }
7424 }
7425 \bool_if:NF \l_@@_transparent_bool
7426 {
7427     \bool_lazy_and:nnF \l_@@_vlines_block_bool \l_@@_hlines_block_bool
7428     {

```

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).

```

7429         \seq_gput_left:Nx \g_@@_pos_of_blocks_seq
7430         { { #1 } { #2 } { #3 } { #4 } { \l_@@_block_name_str } }
7431     }
7432 }

7433 \tl_if_empty:NF \l_@@_draw_tl
7434 {
7435     \bool_lazy_or:nnT \l_@@_hlines_block_bool \l_@@_vlines_block_bool
7436     { \@@_error:n { hlines-with-color } }
7437 }

7438 \tl_if_empty:NF \l_@@_draw_tl
7439 {
7440     \tl_gput_right:Nx \g_nicematrix_code_after_tl
7441     {
7442         \@@_stroke_block:nnn
7443         { \exp_not:n { #5 } } % #5 are the options
7444         { #1 - #2 }
7445         { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7446     }
7447     \seq_gput_right:Nn \g_@@_pos_of_stroken_blocks_seq
7448     { { #1 } { #2 } { #3 } { #4 } }
7449 }

7450 \clist_if_empty:NF \l_@@_borders_clist
7451 {
7452     \tl_gput_right:Nx \g_nicematrix_code_after_tl
7453     {
7454         \@@_stroke_borders_block:nnn
7455         { \exp_not:n { #5 } }
7456         { #1 - #2 }
7457         { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7458     }
7459 }

7460 \tl_if_empty:NF \l_@@_fill_tl
7461 {
7462     \tl_if_empty:NF \l_@@_opacity_tl
7463     {
7464         \tl_if_head_eq_meaning:nNTF \l_@@_fill_tl [
7465             {
7466                 \tl_set:Nx \l_@@_fill_tl
7467                 {
7468                     [ opacity = \l_@@_opacity_tl ,
7469                     \tl_tail:o \l_@@_fill_tl
7470                 }
7471             }
7472             {
7473                 \tl_set:Nx \l_@@_fill_tl
7474                 { [ opacity = \l_@@_opacity_tl ] { \l_@@_fill_tl } }
7475             }

```

```

7476     }
7477     \tl_gput_right:Nx \g_@@_pre_code_before_tl
7478     {
7479         \exp_not:N \roundedrectanglecolor
7480         \exp_args:No \tl_if_head_eq_meaning:nNTF \l_@@_fill_tl [
7481             { \l_@@_fill_tl }
7482             { { \l_@@_fill_tl } }
7483             { #1 - #2 }
7484             { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7485             { \dim_use:N \l_@@_rounded_corners_dim }
7486         }
7487     }
7488     \seq_if_empty:NF \l_@@_tikz_seq
7489     {
7490         \tl_gput_right:Nx \g_nicematrix_code_before_tl
7491         {
7492             \@@_block_tikz:nnnnn
7493             { #1 }
7494             { #2 }
7495             { \int_use:N \l_@@_last_row_int }
7496             { \int_use:N \l_@@_last_col_int }
7497             { \seq_use:Nn \l_@@_tikz_seq { , } }
7498         }
7499     }

7500     \cs_set_protected_nopar:Npn \diagbox ##1 ##2
7501     {
7502         \tl_gput_right:Nx \g_@@_pre_code_after_tl
7503         {
7504             \@@_actually_diagbox:nnnnnn
7505             { #1 }
7506             { #2 }
7507             { \int_use:N \l_@@_last_row_int }
7508             { \int_use:N \l_@@_last_col_int }
7509             { \exp_not:n { ##1 } } { \exp_not:n { ##2 } }
7510         }
7511     }

7512     \hbox_set:Nn \l_@@_cell_box { \set@color #6 }
7513     \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:

```

Let's consider the following `{NiceTabular}`. Because of the instruction `!\hspace{1cm}` in the preamble which increases the space between the columns (by adding, in fact, that space to the previous column, that is to say the second column of the tabular), we will create *two* nodes relative to the block: the node `1-1-block` and the node `1-1-block-short`.

```

\begin{NiceTabular}{cc!\hspace{1cm}}c}
\Block{2-2}{our block} & & one & \\\
& & two & \\\
three & four & five & \\\
six & seven & eight & \\\
\end{NiceTabular}

```

We highlight the node `1-1-block`

our block		one
three	four	two
six	seven	five
		eight

We highlight the node `1-1-block-short`

our block		one
three	four	two
six	seven	five
		eight

The construction of the node corresponding to the merged cells.

```

7514 \pgfpicture
7515 \pgfrememberpicturepositiononpagetrue
7516 \pgf@relevantforpicturesizefalse
7517 \@@_qpoint:n { row - #1 }
7518 \dim_set_eq:NN \l_tmpa_dim \pgf@y
7519 \@@_qpoint:n { col - #2 }
7520 \dim_set_eq:NN \l_tmpb_dim \pgf@x
7521 \@@_qpoint:n { row - \int_eval:n { \l_@@_last_row_int + 1 } }
7522 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7523 \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7524 \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.

```

7525 \@@_pgf_rect_node:nnnnn
7526 { \@@_env: - #1 - #2 - block }
7527 \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
7528 \str_if_empty:NF \l_@@_block_name_str
7529 {
7530 \pgfnodealias
7531 { \@@_env: - \l_@@_block_name_str }
7532 { \@@_env: - #1 - #2 - block }
7533 \str_if_empty:NF \l_@@_name_str
7534 {
7535 \pgfnodealias
7536 { \l_@@_name_str - \l_@@_block_name_str }
7537 { \@@_env: - #1 - #2 - block }
7538 }
7539 }

```

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.

```

7540 \bool_if:NF \l_@@_hpos_of_block_cap_bool
7541 {
7542 \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.

```

7543 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
7544 {

```

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.

```

7545 \cs_if_exist:cT
7546 { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
7547 {
7548 \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
7549 {
7550 \pgfpointanchor { \@@_env: - ##1 - #2 } { west }
7551 \dim_set:Nn \l_tmpb_dim { \dim_min:nn \l_tmpb_dim \pgf@x }
7552 }
7553 }
7554 }

```

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.

```

7555 \dim_compare:nNnT \l_tmpb_dim = \c_max_dim
7556 {
7557 \@@_qpoint:n { col - #2 }
7558 \dim_set_eq:NN \l_tmpb_dim \pgf@x
7559 }

```

```

7560 \dim_set:Nn \l_@@_tmpd_dim { - \c_max_dim }
7561 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
7562 {
7563   \cs_if_exist:cT
7564   { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
7565   {
7566     \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
7567     {
7568       \pgfpointanchor
7569       { \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
7570       { east }
7571       \dim_set:Nn \l_@@_tmpd_dim { \dim_max:nn \l_@@_tmpd_dim \pgf@x }
7572     }
7573   }
7574 }
7575 \dim_compare:nNnT \l_@@_tmpd_dim = { - \c_max_dim }
7576 {
7577   \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7578   \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
7579 }
7580 \@@_pgf_rect_node:nnnnn
7581 { \@@_env: - #1 - #2 - block - short }
7582 \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
7583 }

```

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.

```

7584 \bool_if:NT \l_@@_medium_nodes_bool
7585 {
7586   \@@_pgf_rect_node:nnn
7587   { \@@_env: - #1 - #2 - block - medium }
7588   { \pgfpointanchor { \@@_env: - #1 - #2 - medium } { north-west } }
7589   {
7590     \pgfpointanchor
7591     { \@@_env:
7592       - \int_use:N \l_@@_last_row_int
7593       - \int_use:N \l_@@_last_col_int - medium
7594     }
7595     { south-east }
7596   }
7597 }

```

Now, we will put the label of the block.

```

7598 \bool_lazy_any:nTF
7599 {
7600   { \str_if_eq_p:on \l_@@_vpos_block_str { c } }
7601   { \str_if_eq_p:on \l_@@_vpos_block_str { T } }
7602   { \str_if_eq_p:on \l_@@_vpos_block_str { B } }
7603 }
7604 {

```

If we are in the first column, we must put the block as if it was with the key `r`.

```

7605 \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`.

```

7606 \bool_if:nT \g_@@_last_col_found_bool
7607 {
7608   \int_compare:nNnT { #2 } = \g_@@_col_total_int
7609   { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_l_str }
7610 }

```

`\l_tmpa_tl` will contain the anchor of the PGF node which will be used.

```

7611     \tl_set:Nx \l_tmpa_tl
7612     {
7613         \str_case:on \l_@@_vpos_block_str
7614         {
7615             c {
7616                 \str_case:on \l_@@_hpos_block_str
7617                 {
7618                     c { center }
7619                     l { west }
7620                     r { east }
7621                 }
7622             }
7623             T {
7624                 \str_case:on \l_@@_hpos_block_str
7625                 {
7626                     c { north }
7627                     l { north~west }
7628                     r { north~east }
7629                 }
7630             }
7631             B {
7632                 \str_case:on \l_@@_hpos_block_str
7633                 {
7634                     c { south}
7635                     l { south~west }
7636                     r { south~east }
7637                 }
7638             }
7639         }
7640     }
7641 }
7642
7643 \pgftransformshift
7644 {
7645     \pgfpointanchor
7646     {
7647         \@@_env: - #1 - #2 - block
7648         \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
7649     }
7650     { \l_tmpa_tl }
7651 }
7652
7653 \pgfset
7654 {
7655     inner~xsep = \c_zero_dim ,
7656     inner~ysep = \c_zero_dim
7657 }
7658 \pgfnode
7659 { rectangle }
7660 { \l_tmpa_tl }
7661 { \box_use_drop:N \l_@@_cell_box } { } { }
7662 }

```

End of the case when `\l_@@_vpos_block_str` is equal to c, T or B. Now, the other cases.

```

7663 {
7664     \pgfextracty \l_tmpa_dim
7665     {
7666         \@@_qpoint:n
7667         {
7668             row - \str_if_eq:onTF \l_@@_vpos_block_str { b } { #3 } { #1 }
7669             - base
7670         }

```

```

7671     }
7672     \dim_sub:Nn \l_tmpa_dim { 0.5 \arrayrulewidth } % added 2023-02-21

```

We retrieve (in \pgf@x) the x -value of the center of the block.

```

7673     \pgfpointanchor
7674     {
7675         \@@_env: - #1 - #2 - block
7676         \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
7677     }
7678     {
7679         \str_case:on \l_@@_hpos_block_str
7680         {
7681             c { center }
7682             l { west }
7683             r { east }
7684         }
7685     }

```

We put the label of the block which has been composed in \l_@@_cell_box.

```

7686     \pgftransformshift { \pgfpoint \pgf@x \l_tmpa_dim }
7687     \pgfset { inner~sep = \c_zero_dim }
7688     \pgfnode
7689     { rectangle }
7690     {
7691         \str_case:on \l_@@_hpos_block_str
7692         {
7693             c { base }
7694             l { base~west }
7695             r { base~east }
7696         }
7697     }
7698     { \box_use_drop:N \l_@@_cell_box } { } { }
7699 }
7700 \endpgfpicture
7701 \group_end:
7702 }

```

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).

```

7703 \cs_new_protected:Npn \@@_stroke_block:nnn #1 #2 #3
7704 {
7705     \group_begin:
7706     \tl_clear:N \l_@@_draw_tl
7707     \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
7708     \keys_set_known:nn { NiceMatrix / BlockStroke } { #1 }
7709     \pgfpicture
7710     \pgfrememberpicturepositiononpagetrue
7711     \pgf@relevantforpicturesizefalse
7712     \tl_if_empty:NF \l_@@_draw_tl
7713     {

```

If the user has used the key `color` of the command `\Block` without value, the color fixed by `\arrayrulecolor` is used.

```

7714         \tl_if_eq:NNTF \l_@@_draw_tl \c_@@_default_tl
7715         { \CT@arc@ }
7716         { \@@_color:o \l_@@_draw_tl }
7717     }
7718     \pgfsetcornersarced
7719     {
7720         \pgfpoint
7721         { \l_@@_rounded_corners_dim }
7722         { \l_@@_rounded_corners_dim }

```

```

7723     }
7724     \@@_cut_on_hyphen:w #2 \q_stop
7725     \int_compare:nNnF \l_tmpa_tl > \c@iRow
7726     {
7727         \int_compare:nNnF \l_tmpb_tl > \c@jCol
7728         {
7729             \@@_qpoint:n { row - \l_tmpa_tl }
7730             \dim_set_eq:NN \l_tmpb_dim \pgf@y
7731             \@@_qpoint:n { col - \l_tmpb_tl }
7732             \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
7733             \@@_cut_on_hyphen:w #3 \q_stop
7734             \int_compare:nNnT \l_tmpa_tl > \c@iRow
7735             { \tl_set:No \l_tmpa_tl { \int_use:N \c@iRow } }
7736             \int_compare:nNnT \l_tmpb_tl > \c@jCol
7737             { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
7738             \@@_qpoint:n { row - \int_eval:n { \l_tmpa_tl + 1 } }
7739             \dim_set_eq:NN \l_tmpa_dim \pgf@y
7740             \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
7741             \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
7742             \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
7743             \pgfpathrectanglecorners
7744             { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
7745             { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
7746             \dim_compare:nNnTF \l_@@_rounded_corners_dim = \c_zero_dim
7747             { \pgfusepathqstroke }
7748             { \pgfusepath { stroke } }
7749         }
7750     }
7751     \endpgfpicture
7752     \group_end:
7753 }

```

Here is the set of keys for the command `\@@_stroke_block:nnn`.

```

7754 \keys_define:nn { NiceMatrix / BlockStroke }
7755 {
7756     color .tl_set:N = \l_@@_draw_tl ,
7757     draw .code:n =
7758         \exp_args:Ne \tl_if_empty:nF { #1 } { \tl_set:Nn \l_@@_draw_tl { #1 } } ,
7759     draw .default:n = default ,
7760     line-width .dim_set:N = \l_@@_line_width_dim ,
7761     rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
7762     rounded-corners .default:n = 4 pt
7763 }

```

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).

```

7764 \cs_new_protected:Npn \@@_vlines_block:nnn #1 #2 #3
7765 {
7766     \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
7767     \keys_set_known:nn { NiceMatrix / BlockBorders } { #1 }
7768     \@@_cut_on_hyphen:w #2 \q_stop
7769     \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
7770     \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
7771     \@@_cut_on_hyphen:w #3 \q_stop
7772     \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
7773     \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
7774     \int_step_inline:nnn \l_@@_tmpd_tl \l_tmpb_tl
7775     {
7776         \use:e
7777         {
7778             \@@_vline:n
7779             {

```

```

7780         position = ##1 ,
7781         start = \l_@@_tmpc_tl ,
7782         end = \int_eval:n { \l_tmpa_tl - 1 } ,
7783         total-width = \dim_use:N \l_@@_line_width_dim
7784     }
7785 }
7786 }
7787 }
7788 \cs_new_protected:Npn \@@_hlines_block:nnn #1 #2 #3
7789 {
7790     \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
7791     \keys_set_known:nn { NiceMatrix / BlockBorders } { #1 }
7792     \@@_cut_on_hyphen:w #2 \q_stop
7793     \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
7794     \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
7795     \@@_cut_on_hyphen:w #3 \q_stop
7796     \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
7797     \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
7798     \int_step_inline:nnn \l_@@_tmpc_tl \l_tmpa_tl
7799     {
7800         \use:e
7801         {
7802             \@@_hline:n
7803             {
7804                 position = ##1 ,
7805                 start = \l_@@_tmpd_tl ,
7806                 end = \int_eval:n { \l_tmpb_tl - 1 } ,
7807                 total-width = \dim_use:N \l_@@_line_width_dim
7808             }
7809         }
7810     }
7811 }

```

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).

```

7812 \cs_new_protected:Npn \@@_stroke_borders_block:nnn #1 #2 #3
7813 {
7814     \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
7815     \keys_set_known:nn { NiceMatrix / BlockBorders } { #1 }
7816     \dim_compare:nNnTF \l_@@_rounded_corners_dim > \c_zero_dim
7817     { \@@_error:n { borders~forbidden } }
7818     {
7819         \tl_clear_new:N \l_@@_borders_tikz_tl
7820         \keys_set:nV
7821         { NiceMatrix / OnlyForTikzInBorders }
7822         \l_@@_borders_clist
7823         \@@_cut_on_hyphen:w #2 \q_stop
7824         \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
7825         \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
7826         \@@_cut_on_hyphen:w #3 \q_stop
7827         \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
7828         \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
7829         \@@_stroke_borders_block_i:
7830     }
7831 }
7832 \hook_gput_code:nnn { begindocument } { . }
7833 {
7834     \cs_new_protected:Npx \@@_stroke_borders_block_i:
7835     {
7836         \c_@@_pgfortikzpicture_tl
7837         \@@_stroke_borders_block_ii:

```

```

7838     \c_@@_endpgfortikzpicture_tl
7839   }
7840 }

7841 \cs_new_protected:Npn \@@_stroke_borders_block_ii:
7842 {
7843   \pgfrememberpicturepositiononpagetrue
7844   \pgf@relevantforpicturesizefalse
7845   \CT@arc@
7846   \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
7847   \clist_if_in:NnT \l_@@_borders_clist { right }
7848     { \@@_stroke_vertical:n \l_tmpb_tl }
7849   \clist_if_in:NnT \l_@@_borders_clist { left }
7850     { \@@_stroke_vertical:n \l_@@_tmpd_tl }
7851   \clist_if_in:NnT \l_@@_borders_clist { bottom }
7852     { \@@_stroke_horizontal:n \l_tmpa_tl }
7853   \clist_if_in:NnT \l_@@_borders_clist { top }
7854     { \@@_stroke_horizontal:n \l_@@_tmpc_tl }
7855 }

7856 \keys_define:nn { NiceMatrix / OnlyForTikzInBorders }
7857 {
7858   tikz .code:n =
7859     \cs_if_exist:NTF \tikzpicture
7860       { \tl_set:Nn \l_@@_borders_tikz_tl { #1 } }
7861       { \@@_error:n { tikz~in~borders~without~tikz } } ,
7862   tikz .value_required:n = true ,
7863   top .code:n = ,
7864   bottom .code:n = ,
7865   left .code:n = ,
7866   right .code:n = ,
7867   unknown .code:n = \@@_error:n { bad~border }
7868 }

```

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).

```

7869 \cs_new_protected:Npn \@@_stroke_vertical:n #1
7870 {
7871   \@@_qpoint:n \l_@@_tmpc_tl
7872   \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
7873   \@@_qpoint:n \l_tmpa_tl
7874   \dim_set:Nn \l_@@_tmpc_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
7875   \@@_qpoint:n { #1 }
7876   \tl_if_empty:NTF \l_@@_borders_tikz_tl
7877   {
7878     \pgfpathmoveto { \pgfpoint \pgf@x \l_tmpb_dim }
7879     \pgfpathlineto { \pgfpoint \pgf@x \l_@@_tmpc_dim }
7880     \pgfusepath{stroke}
7881   }
7882   {
7883     \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
7884     ( \pgf@x , \l_tmpb_dim ) -- ( \pgf@x , \l_@@_tmpc_dim ) ;
7885   }
7886 }

```

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).

```

7887 \cs_new_protected:Npn \@@_stroke_horizontal:n #1
7888 {
7889   \@@_qpoint:n \l_@@_tmpd_tl
7890   \clist_if_in:NnTF \l_@@_borders_clist { left }
7891     { \dim_set:Nn \l_tmpa_dim { \pgf@x - 0.5 \l_@@_line_width_dim } }
7892     { \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \l_@@_line_width_dim } }
7893   \@@_qpoint:n \l_tmpb_tl

```

```

7894 \dim_set:Nn \l_tmpb_dim { \pgf@x + 0.5 \l_@@_line_width_dim }
7895 \@@_qpoint:n { #1 }
7896 \tl_if_empty:NTF \l_@@_borders_tikz_tl
7897 {
7898   \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }
7899   \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
7900   \pgfusepathqstroke
7901 }
7902 {
7903   \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
7904   ( \l_tmpa_dim , \pgf@y ) -- ( \l_tmpb_dim , \pgf@y ) ;
7905 }
7906 }

```

Here is the set of keys for the command `\@@_stroke_borders_block:nnn`.

```

7907 \keys_define:nn { NiceMatrix / BlockBorders }
7908 {
7909   borders .clist_set:N = \l_@@_borders_clist ,
7910   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
7911   rounded-corners .default:n = 4 pt ,
7912   line-width .dim_set:N = \l_@@_line_width_dim
7913 }

```

The following command will be used if the key `tikz` has been used for the command `\Block`. The arguments `#1` and `#2` are the coordinates of the first cell and `#3` and `#4` the coordinates of the last cell of the block. `#5` is a comma-separated list of the Tikz keys used with the path. However, among those keys, you have added in `nicematrix` a special key `offset` (an offset for the rectangle of the block). That's why we have to extract that key first.

```

7914 \cs_new_protected:Npn \@@_block_tikz:nnnnn #1 #2 #3 #4 #5
7915 {
7916   \begin { tikzpicture }
7917   \@@_clip_with_rounded_corners:
7918   \clist_map_inline:nn { #5 }
7919   {
7920     \keys_set_known:nnN { NiceMatrix / SpecialOffset } { ##1 } \l_tmpa_tl
7921     \use:e { \exp_not:N \path [ \l_tmpa_tl ] }
7922     (
7923       [
7924         xshift = \dim_use:N \l_@@_offset_dim ,
7925         yshift = - \dim_use:N \l_@@_offset_dim
7926       ]
7927       #1 -| #2
7928     )
7929     rectangle
7930     (
7931       [
7932         xshift = - \dim_use:N \l_@@_offset_dim ,
7933         yshift = \dim_use:N \l_@@_offset_dim
7934       ]
7935       \int_eval:n { #3 + 1 } -| \int_eval:n { #4 + 1 }
7936     ) ;
7937   }
7938   \end { tikzpicture }
7939 }
7940 \cs_generate_variant:Nn \@@_block_tikz:nnnnn { n n n n V }

7941 \keys_define:nn { NiceMatrix / SpecialOffset }
7942 { offset .dim_set:N = \l_@@_offset_dim }

```

28 How to draw the dotted lines transparently

```

7943 \cs_set_protected:Npn \@@_renew_matrix:
7944 {
7945   \RenewDocumentEnvironment { pmatrix } { } {
7946     { \pNiceMatrix }
7947     { \endpNiceMatrix }
7948   \RenewDocumentEnvironment { vmatrix } { } {
7949     { \vNiceMatrix }
7950     { \endvNiceMatrix }
7951   \RenewDocumentEnvironment { Vmatrix } { } {
7952     { \VNiceMatrix }
7953     { \endVNiceMatrix }
7954   \RenewDocumentEnvironment { bmatrix } { } {
7955     { \bNiceMatrix }
7956     { \endbNiceMatrix }
7957   \RenewDocumentEnvironment { Bmatrix } { } {
7958     { \BNiceMatrix }
7959     { \endBNiceMatrix }
7960 }

```

29 Automatic arrays

We will extract some keys and pass the other keys to the environment `{NiceArrayWithDelims}`.

```

7961 \keys_define:nn { NiceMatrix / Auto }
7962 {
7963   columns-type .tl_set:N = \l_@@_columns_type_tl ,
7964   columns-type .value_required:n = true ,
7965   l .meta:n = { columns-type = l } ,
7966   r .meta:n = { columns-type = r } ,
7967   c .meta:n = { columns-type = c } ,
7968   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
7969   delimiters / color .value_required:n = true ,
7970   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
7971   delimiters / max-width .default:n = true ,
7972   delimiters .code:n = \keys_set:nn { NiceMatrix / delimiters } { #1 } ,
7973   delimiters .value_required:n = true ,
7974   rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
7975   rounded-corners .default:n = 4 pt
7976 }
7977 \NewDocumentCommand \AutoNiceMatrixWithDelims
7978 { m m O { } } > { \SplitArgument { 1 } { - } } m O { } m ! O { } }
7979 { \@@_auto_nice_matrix:nnnnnn { #1 } { #2 } #4 { #6 } { #3 , #5 , #7 } }
7980 \cs_new_protected:Npn \@@_auto_nice_matrix:nnnnnn #1 #2 #3 #4 #5 #6
7981 {

```

The group is for the protection of the keys.

```

7982 \group_begin:
7983 \keys_set_known:nnN { NiceMatrix / Auto } { #6 } \l_tmpa_tl
7984 \use:e
7985 {
7986   \exp_not:N \begin { NiceArrayWithDelims } { #1 } { #2 }
7987     { * { #4 } { \exp_not:o \l_@@_columns_type_tl } }
7988     [ \exp_not:o \l_tmpa_tl ]
7989 }
7990 \int_if_zero:nT \l_@@_first_row_int
7991 {
7992   \int_if_zero:nT \l_@@_first_col_int { & }
7993   \prg_replicate:nn { #4 - 1 } { & }
7994   \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \

```

```

7995     }
7996     \prg_replicate:nn { #3 }
7997     {
7998         \int_if_zero:nT \l_@@_first_col_int { & }

```

We put { } before #6 to avoid a hasty expansion of a potential \arabic{iRow} at the beginning of the row which would result in an incorrect value of that iRow (since iRow is incremented in the first cell of the row of the \halign).

```

7999         \prg_replicate:nn { #4 - 1 } { { } #5 & } #5
8000         \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
8001     }
8002     \int_compare:nNnT \l_@@_last_row_int > { -2 }
8003     {
8004         \int_if_zero:nT \l_@@_first_col_int { & }
8005         \prg_replicate:nn { #4 - 1 } { & }
8006         \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
8007     }
8008     \end { NiceArrayWithDelims }
8009     \group_end:
8010 }

```

```

8011 \cs_set_protected:Npn \@@_define_com:nnn #1 #2 #3
8012 {
8013     \cs_set_protected:cpn { #1 AutoNiceMatrix }
8014     {
8015         \bool_gset_true:N \g_@@_delims_bool
8016         \str_gset:Nx \g_@@_name_env_str { #1 AutoNiceMatrix }
8017         \AutoNiceMatrixWithDelims { #2 } { #3 }
8018     }
8019 }

```

```

8020 \@@_define_com:nnn p ( )
8021 \@@_define_com:nnn b [ ]
8022 \@@_define_com:nnn v | |
8023 \@@_define_com:nnn V \ | \ |
8024 \@@_define_com:nnn B \{ \}

```

We define also a command \AutoNiceMatrix similar to the environment {NiceMatrix}.

```

8025 \NewDocumentCommand \AutoNiceMatrix { O { } m O { } m ! O { } }
8026 {
8027     \group_begin:
8028     \bool_gset_false:N \g_@@_delims_bool
8029     \AutoNiceMatrixWithDelims . . { #2 } { #4 } [ #1 , #3 , #5 ]
8030     \group_end:
8031 }

```

30 The redefinition of the command \dotfill

```

8032 \cs_set_eq:NN \@@_old_dotfill \dotfill
8033 \cs_new_protected:Npn \@@_dotfill:
8034 {

```

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}).

```

8035     \@@_old_dotfill
8036     \tl_gput_right:Nn \g_@@_cell_after_hook_tl \@@_dotfill_i:
8037 }

```

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.

```

8038 \cs_new_protected:Npn \@@_dotfill_i:
8039 { \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } = \c_zero_dim \@@_old_dotfill }

```

31 The command `\diagbox`

The command `\diagbox` will be linked to `\diagbox:nn` in the environments of `nicematrix`. However, there are also redefinitions of `\diagbox` in other circumstances.

```

8040 \cs_new_protected:Npn \@@_diagbox:nn #1 #2
8041 {
8042   \tl_gput_right:Nx \g_@@_pre_code_after_tl
8043   {
8044     \@@_actually_diagbox:nnnnnn
8045     { \int_use:N \c@iRow }
8046     { \int_use:N \c@jCol }
8047     { \int_use:N \c@iRow }
8048     { \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.

```

8049   { \g_@@_row_style_tl \exp_not:n { #1 } }
8050   { \g_@@_row_style_tl \exp_not:n { #2 } }
8051 }

```

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`.

```

8052   \seq_gput_right:Nx \g_@@_pos_of_blocks_seq
8053   {
8054     { \int_use:N \c@iRow }
8055     { \int_use:N \c@jCol }
8056     { \int_use:N \c@iRow }
8057     { \int_use:N \c@jCol }

```

The last argument is for the name of the block.

```

8058     { }
8059   }
8060 }

```

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.

```

8061 \cs_new_protected:Npn \@@_actually_diagbox:nnnnnn #1 #2 #3 #4 #5 #6
8062 {
8063   \pgfpicture
8064   \pgf@relevantforpicturesizefalse
8065   \pgfrememberpicturepositiononpagetrue
8066   \@@_qpoint:n { row - #1 }
8067   \dim_set_eq:NN \l_tmpa_dim \pgf@y
8068   \@@_qpoint:n { col - #2 }
8069   \dim_set_eq:NN \l_tmpb_dim \pgf@x
8070   \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
8071   \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
8072   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8073   \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
8074   \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
8075   \pgfpathlineto { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
8076   {

```

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.

```

8077     \CT@arc@
8078     \pgfsetroundcap
8079     \pgfusepathqstroke
8080   }
8081   \pgfset { inner~sep = 1 pt }
8082   \pgfscope
8083   \pgftransformshift { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
8084   \pgfnode { rectangle } { south~west }
8085   {
8086     \begin { minipage } { 20 cm }
8087     \@@_math_toggle: #5 \@@_math_toggle:
8088     \end { minipage }
8089   }
8090   { }
8091   { }
8092   \endpgfscope
8093   \pgftransformshift { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
8094   \pgfnode { rectangle } { north~east }
8095   {
8096     \begin { minipage } { 20 cm }
8097     \raggedleft
8098     \@@_math_toggle: #6 \@@_math_toggle:
8099     \end { minipage }
8100   }
8101   { }
8102   { }
8103   \endpgfpicture
8104 }

```

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. 82.

In the environments of `nicematrix`, `\CodeAfter` will be linked to `\@@_CodeAfter:`. That macro must *not* be protected since it begins with `\omit`.

```

8105 \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 `\`.

```

8106 \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`.

```

8107 \cs_new_protected:Npn \@@_CodeAfter_ii:n #1 \end
8108 {
8109   \tl_gput_right:Nn \g_nicematrix_code_after_tl { #1 }
8110   \@@_CodeAfter_iv:n
8111 }

```

We catch the argument of the command `\end` (in #1).

```

8112 \cs_new_protected:Npn \@@_CodeAfter_iv:n #1
8113 {

```

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.

```
8114 \str_if_eq:eeTF \@currenvir { #1 }
8115 { \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`.

```
8116 {
8117   \tl_gput_right:Nn \g_nicematrix_code_after_tl { \end { #1 } }
8118   \@@_CodeAfter_ii:n
8119 }
8120 }
```

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.

```
8121 \cs_new_protected:Npn \@@_delimiter:nnn #1 #2 #3
8122 {
8123   \pgfpicture
8124   \pgfrememberpicturepositiononpagetrue
8125   \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.

```
8126 \@@_qpoint:n { row - 1 }
8127 \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
8128 \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
8129 \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).

```
8130 \bool_if:nTF { #3 }
8131 { \dim_set_eq:NN \l_tmpa_dim \c_max_dim }
8132 { \dim_set:Nn \l_tmpa_dim { - \c_max_dim } }
8133 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
8134 {
8135   \cs_if_exist:cT
8136   { pgf @ sh @ ns @ \@_env: - ##1 - #2 }
8137   {
8138     \pgfpictureanchor
8139     { \@_env: - ##1 - #2 }
8140     { \bool_if:nTF { #3 } { west } { east } }
8141     \dim_set:Nn \l_tmpa_dim
8142     { \bool_if:nTF { #3 } \dim_min:nn \dim_max:nn \l_tmpa_dim \pgf@x }
8143   }
8144 }
```

Now we can put the delimiter with a node of PGF.

```

8145 \pgfset { inner~sep = \c_zero_dim }
8146 \dim_zero:N \nulldelimiterspace
8147 \pgftransformshift
8148 {
8149   \pgfpoint
8150   { \l_tmpa_dim }
8151   { ( \l_@@_y_initial_dim + \l_@@_y_final_dim + \arrayrulewidth ) / 2 }
8152 }
8153 \pgfnode
8154 { rectangle }
8155 { \bool_if:nTF { #3 } { east } { west } }
8156 {

```

Here is the content of the PGF node, that is to say the delimiter, constructed with its right size.

```

8157 \nullfont
8158 \c_math_toggle_token
8159 \@@_color:o \l_@@_delimiters_color_tl
8160 \bool_if:nTF { #3 } { \left #1 } { \left . }
8161 \vcenter
8162 {
8163   \nullfont
8164   \hrule \@height
8165   \dim_eval:n { \l_@@_y_initial_dim - \l_@@_y_final_dim }
8166   \@depth \c_zero_dim
8167   \@width \c_zero_dim
8168 }
8169 \bool_if:nTF { #3 } { \right . } { \right #1 }
8170 \c_math_toggle_token
8171 }
8172 { }
8173 { }
8174 \endpgfpicture
8175 }

```

34 The command \SubMatrix

```

8176 \keys_define:nn { NiceMatrix / sub-matrix }
8177 {
8178   extra-height .dim_set:N = \l_@@_submatrix_extra_height_dim ,
8179   extra-height .value_required:n = true ,
8180   left-xshift .dim_set:N = \l_@@_submatrix_left_xshift_dim ,
8181   left-xshift .value_required:n = true ,
8182   right-xshift .dim_set:N = \l_@@_submatrix_right_xshift_dim ,
8183   right-xshift .value_required:n = true ,
8184   xshift .meta:n = { left-xshift = #1, right-xshift = #1 } ,
8185   xshift .value_required:n = true ,
8186   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8187   delimiters / color .value_required:n = true ,
8188   slim .bool_set:N = \l_@@_submatrix_slim_bool ,
8189   slim .default:n = true ,
8190   hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
8191   hlines .default:n = all ,
8192   vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
8193   vlines .default:n = all ,
8194   hvlines .meta:n = { hlines, vlines } ,
8195   hvlines .value_forbidden:n = true
8196 }
8197 \keys_define:nn { NiceMatrix }
8198 {
8199   SubMatrix .inherit:n = NiceMatrix / sub-matrix ,

```

```

8200 NiceArray / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
8201 pNiceArray / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
8202 NiceMatrixOptions / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
8203 }

```

The following keys set is for the command \SubMatrix itself (not the tuning of \SubMatrix that can be done elsewhere).

```

8204 \keys_define:nn { NiceMatrix / SubMatrix }
8205 {
8206   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8207   delimiters / color .value_required:n = true ,
8208   hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
8209   hlines .default:n = all ,
8210   vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
8211   vlines .default:n = all ,
8212   hvlines .meta:n = { hlines, vlines } ,
8213   hvlines .value_forbidden:n = true ,
8214   name .code:n =
8215     \tl_if_empty:nTF { #1 }
8216     { \@@_error:n { Invalid-name } }
8217     {
8218       \regex_match:nnTF { \A[A-Za-z][A-Za-z0-9]*\Z } { #1 }
8219       {
8220         \seq_if_in:NnTF \g_@@_submatrix_names_seq { #1 }
8221         { \@@_error:nn { Duplicate-name-for-SubMatrix } { #1 } }
8222         {
8223           \str_set:Nn \l_@@_submatrix_name_str { #1 }
8224           \seq_gput_right:Nn \g_@@_submatrix_names_seq { #1 }
8225         }
8226       }
8227       { \@@_error:n { Invalid-name } }
8228     } ,
8229   name .value_required:n = true ,
8230   rules .code:n = \keys_set:nn { NiceMatrix / rules } { #1 } ,
8231   rules .value_required:n = true ,
8232   code .tl_set:N = \l_@@_code_tl ,
8233   code .value_required:n = true ,
8234   unknown .code:n = \@@_error:n { Unknown-key-for-SubMatrix }
8235 }

8236 \NewDocumentCommand \@@_SubMatrix_in_code_before { m m m m ! O { } }
8237 {
8238   \peek_remove_spaces:n
8239   {
8240     \tl_gput_right:Nx \g_@@_pre_code_after_tl
8241     {
8242       \SubMatrix { #1 } { #2 } { #3 } { #4 }
8243       [
8244         delimiters / color = \l_@@_delimiters_color_tl ,
8245         hlines = \l_@@_submatrix_hlines_clist ,
8246         vlines = \l_@@_submatrix_vlines_clist ,
8247         extra-height = \dim_use:N \l_@@_submatrix_extra_height_dim ,
8248         left-xshift = \dim_use:N \l_@@_submatrix_left_xshift_dim ,
8249         right-xshift = \dim_use:N \l_@@_submatrix_right_xshift_dim ,
8250         slim = \bool_to_str:N \l_@@_submatrix_slim_bool ,
8251         #5
8252       ]
8253     }
8254     \@@_SubMatrix_in_code_before_i { #2 } { #3 }
8255   }
8256 }

8257 \NewDocumentCommand \@@_SubMatrix_in_code_before_i
8258 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }

```

```

8259 { \@@_SubMatrix_in_code_before_i:nnnn #1 #2 }
8260 \cs_new_protected:Npn \@@_SubMatrix_in_code_before_i:nnnn #1 #2 #3 #4
8261 {
8262   \seq_gput_right:Nx \g_@@_submatrix_seq
8263   {

```

We use `\str_if_eq:nnTF` because it is fully expandable.

```

8264     { \str_if_eq:nnTF { #1 } { last } { \int_use:N \c@iRow } { #1 } }
8265     { \str_if_eq:nnTF { #2 } { last } { \int_use:N \c@jCol } { #2 } }
8266     { \str_if_eq:nnTF { #3 } { last } { \int_use:N \c@iRow } { #3 } }
8267     { \str_if_eq:nnTF { #4 } { last } { \int_use:N \c@jCol } { #4 } }
8268   }
8269 }

```

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`.

```

8270 \hook_gput_code:nnn { begindocument } { . }
8271 {
8272   \cs_set_nopar:Npn \l_@@_argspec_tl { m m m m 0 { } E { _ ^ } { { } { } } }
8273   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
8274   \exp_args:NNo \NewDocumentCommand \@@_SubMatrix \l_@@_argspec_tl
8275   {
8276     \peek_remove_spaces:n
8277     {
8278       \@@_sub_matrix:nnnnnnn
8279       { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } { #7 }
8280     }
8281   }
8282 }

```

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).

```

8283 \NewDocumentCommand \@@_compute_i_j:nn
8284 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
8285 { \@@_compute_i_j:nnnn #1 #2 }
8286 \cs_new_protected:Npn \@@_compute_i_j:nnnn #1 #2 #3 #4
8287 {
8288   \cs_set_nopar:Npn \l_@@_first_i_tl { #1 }
8289   \cs_set_nopar:Npn \l_@@_first_j_tl { #2 }
8290   \cs_set_nopar:Npn \l_@@_last_i_tl { #3 }
8291   \cs_set_nopar:Npn \l_@@_last_j_tl { #4 }
8292   \tl_if_eq:NnT \l_@@_first_i_tl { last }
8293   { \tl_set:NV \l_@@_first_i_tl \c@iRow }
8294   \tl_if_eq:NnT \l_@@_first_j_tl { last }
8295   { \tl_set:NV \l_@@_first_j_tl \c@jCol }
8296   \tl_if_eq:NnT \l_@@_last_i_tl { last }

```

```

8297     { \tl_set:NV \l_@@_last_i_tl \c@iRow }
8298     \tl_if_eq:NnT \l_@@_last_j_tl { last }
8299     { \tl_set:NV \l_@@_last_j_tl \c@jCol }
8300   }

8301   \cs_new_protected:Npn \@@_sub_matrix:nnnnnnn #1 #2 #3 #4 #5 #6 #7
8302   {
8303     \group_begin:

```

The four following token lists correspond to the position of the \SubMatrix.

```

8304     \@@_compute_i_j:nn { #2 } { #3 }
8305     \int_compare:nNtT \l_@@_first_i_tl = \l_@@_last_i_tl
8306     { \cs_set_nopar:Npn \arraystretch { 1 } }
8307     \bool_lazy_or:nnTF
8308     { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
8309     { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
8310     { \@@_error:nn { Construct-too-large } { \SubMatrix } }
8311     {
8312       \str_clear_new:N \l_@@_submatrix_name_str
8313       \keys_set:nn { NiceMatrix / SubMatrix } { #5 }
8314       \pgfpicture
8315       \pgfrememberpicturepositiononpagetrue
8316       \pgf@relevantforpicturesizefalse
8317       \pgfset { inner~sep = \c_zero_dim }
8318       \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
8319       \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }

```

The last value of \int_step_inline:nnn is provided by currying.

```

8320     \bool_if:NtF \l_@@_submatrix_slim_bool
8321     { \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl }
8322     { \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int }
8323     {
8324       \cs_if_exist:cT
8325       { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
8326       {
8327         \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
8328         \dim_set:Nn \l_@@_x_initial_dim
8329         { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
8330       }
8331       \cs_if_exist:cT
8332       { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
8333       {
8334         \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
8335         \dim_set:Nn \l_@@_x_final_dim
8336         { \dim_max:nn \l_@@_x_final_dim \pgf@x }
8337       }
8338     }
8339     \dim_compare:nNtF \l_@@_x_initial_dim = \c_max_dim
8340     { \@@_error:nn { Impossible-delimiter } { left } }
8341     {
8342       \dim_compare:nNtF \l_@@_x_final_dim = { - \c_max_dim }
8343       { \@@_error:nn { Impossible-delimiter } { right } }
8344       { \@@_sub_matrix_i:nnnn { #1 } { #4 } { #6 } { #7 } }
8345     }
8346     \endpgfpicture
8347   }
8348   \group_end:
8349 }

```

#1 is the left delimiter, #2 is the right one, #3 is the subscript and #4 is the superscript.

```

8350   \cs_new_protected:Npn \@@_sub_matrix_i:nnnn #1 #2 #3 #4
8351   {
8352     \@@_qpoint:n { row - \l_@@_first_i_tl - base }
8353     \dim_set:Nn \l_@@_y_initial_dim

```

```

8354 {
8355   \fp_to_dim:n
8356   {
8357     \pgf@y
8358     + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
8359   }
8360 }
8361 \@@_qpoint:n { row - \l_@@_last_i_tl - base }
8362 \dim_set:Nn \l_@@_y_final_dim
8363 { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
8364 \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
8365 {
8366   \cs_if_exist:cT
8367   { pgf @ sh @ ns @ \@@_env: - \l_@@_first_i_tl - ##1 }
8368   {
8369     \pgfpointanchor { \@@_env: - \l_@@_first_i_tl - ##1 } { north }
8370     \dim_set:Nn \l_@@_y_initial_dim
8371     { \dim_max:nn \l_@@_y_initial_dim \pgf@y }
8372   }
8373   \cs_if_exist:cT
8374   { pgf @ sh @ ns @ \@@_env: - \l_@@_last_i_tl - ##1 }
8375   {
8376     \pgfpointanchor { \@@_env: - \l_@@_last_i_tl - ##1 } { south }
8377     \dim_set:Nn \l_@@_y_final_dim
8378     { \dim_min:nn \l_@@_y_final_dim \pgf@y }
8379   }
8380 }
8381 \dim_set:Nn \l_tmpa_dim
8382 {
8383   \l_@@_y_initial_dim - \l_@@_y_final_dim +
8384   \l_@@_submatrix_extra_height_dim - \arrayrulewidth
8385 }
8386 \dim_zero:N \nulldelimiterspace

```

We will draw the rules in the `\SubMatrix`.

```

8387 \group_begin:
8388 \pgfsetlinewidth { 1.1 \arrayrulewidth }
8389 \@@_set_CT@arc@:o \l_@@_rules_color_tl
8390 \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`.

```

8391 \seq_map_inline:Nn \g_@@_cols_vlism_seq
8392 {
8393   \int_compare:nNnT \l_@@_first_j_tl < { ##1 }
8394   {
8395     \int_compare:nNnT
8396     { ##1 } < { \int_eval:n { \l_@@_last_j_tl + 1 } }
8397     {

```

First, we extract the value of the abscissa of the rule we have to draw.

```

8398 \@@_qpoint:n { col - ##1 }
8399 \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
8400 \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
8401 \pgfusepathqstroke
8402 }
8403 }
8404 }

```

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.

```

8405 \tl_if_eq:NNTF \l_@@_submatrix_vlines_clist \c_@@_all_tl

```

```

8406 { \int_step_inline:nn { \l_@@_last_j_tl - \l_@@_first_j_tl } }
8407 { \clist_map_inline:Nn \l_@@_submatrix_vlines_clist }
8408 {
8409   \bool_lazy_and:nnTF
8410     { \int_compare_p:nNn { ##1 } > \c_zero_int }
8411     {
8412       \int_compare_p:nNn
8413         { ##1 } < { \l_@@_last_j_tl - \l_@@_first_j_tl + 1 } }
8414     {
8415       \@@_qpoint:n { col - \int_eval:n { ##1 + \l_@@_first_j_tl } }
8416       \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
8417       \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
8418       \pgfusepathqstroke
8419     }
8420     { \@@_error:nmn { Wrong~line~in~SubMatrix } { vertical } { ##1 } }
8421 }

```

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.

```

8422 \tl_if_eq:NNTF \l_@@_submatrix_hlines_clist \c_@@_all_tl
8423 { \int_step_inline:nn { \l_@@_last_i_tl - \l_@@_first_i_tl } }
8424 { \clist_map_inline:Nn \l_@@_submatrix_hlines_clist }
8425 {
8426   \bool_lazy_and:nnTF
8427     { \int_compare_p:nNn { ##1 } > \c_zero_int }
8428     {
8429       \int_compare_p:nNn
8430         { ##1 } < { \l_@@_last_i_tl - \l_@@_first_i_tl + 1 } }
8431     {
8432       \@@_qpoint:n { row - \int_eval:n { ##1 + \l_@@_first_i_tl } }

```

We use a group to protect `\l_tmpa_dim` and `\l_tmpb_dim`.

```

8433   \group_begin:
We compute in \l_tmpa_dim the x-value of the left end of the rule.
8434   \dim_set:Nn \l_tmpa_dim
8435     { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
8436   \str_case:nn { #1 }
8437   {
8438     ( { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
8439     [ { \dim_sub:Nn \l_tmpa_dim { 0.2 mm } }
8440     \{ { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
8441     }
8442   \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }

```

We compute in `\l_tmpb_dim` the *x*-value of the right end of the rule.

```

8443   \dim_set:Nn \l_tmpb_dim
8444     { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
8445   \str_case:nn { #2 }
8446   {
8447     ) { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
8448     ] { \dim_add:Nn \l_tmpb_dim { 0.2 mm } }
8449     \} { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
8450   }
8451   \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
8452   \pgfusepathqstroke
8453   \group_end:
8454 }
8455 { \@@_error:nmn { Wrong~line~in~SubMatrix } { horizontal } { ##1 } }
8456 }

```

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).

```

8457 \str_if_empty:NF \l_@@_submatrix_name_str

```

```

8458     {
8459         \@@_pgf_rect_node:nnnnn \l_@@_submatrix_name_str
8460         \l_@@_x_initial_dim \l_@@_y_initial_dim
8461         \l_@@_x_final_dim \l_@@_y_final_dim
8462     }
8463     \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`.

```

8464     \begin { pgfscope }
8465     \pgftransformshift
8466     {
8467         \pgfpoint
8468         { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
8469         { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
8470     }
8471     \str_if_empty:NTF \l_@@_submatrix_name_str
8472     { \@@_node_left:nn #1 { } }
8473     { \@@_node_left:nn #1 { \@@_env: - \l_@@_submatrix_name_str - left } }
8474     \end { pgfscope }

```

Now, we deal with the right delimiter.

```

8475     \pgftransformshift
8476     {
8477         \pgfpoint
8478         { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
8479         { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
8480     }
8481     \str_if_empty:NTF \l_@@_submatrix_name_str
8482     { \@@_node_right:nnnn #2 { } { #3 } { #4 } }
8483     {
8484         \@@_node_right:nnnn #2
8485         { \@@_env: - \l_@@_submatrix_name_str - right } { #3 } { #4 }
8486     }
8487     \cs_set_eq:NN \pgfpointanchor \@@_pgfpointanchor:n
8488     \flag_clear_new:n { nicematrix }
8489     \l_@@_code_tl
8490 }

```

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`.

```

8491 \cs_set_eq:NN \@@_old_pgfpointanchor \pgfpointanchor

```

The following command will be linked to `\pgfpointanchor` just before the execution of the option code of the command `\SubMatrix`. In this command, we catch the argument `#1` of `\pgfpointanchor` and we apply to it the command `\@@_pgfpointanchor_i:nn` before passing it to the original `\pgfpointanchor`. We have to act in an expandable way because the command `\pgfpointanchor` is used in names of Tikz nodes which are computed in an expandable way.

```

8492 \cs_new_protected:Npn \@@_pgfpointanchor:n #1
8493 {
8494     \use:e
8495     { \exp_not:N \@@_old_pgfpointanchor { \@@_pgfpointanchor_i:nn #1 } }
8496 }

```

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 `-`.

```

8497 \cs_new:Npn \@@_pgfpointanchor_i:nn #1 #2
8498 { #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.

```

8499 \tl_const:Nn \c_@@_integers_alist_tl
8500 {
8501   { 1 } { } { 2 } { } { 3 } { } { 4 } { } { 5 } { }
8502   { 6 } { } { 7 } { } { 8 } { } { 9 } { } { 10 } { }
8503   { 11 } { } { 12 } { } { 13 } { } { 14 } { } { 15 } { }
8504   { 16 } { } { 17 } { } { 18 } { } { 19 } { } { 20 } { }
8505 }

```

```

8506 \cs_new:Npn \@@_pgfpointanchor_ii:w #1-#2\q_stop
8507 {

```

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`.

```

8508   \tl_if_empty:nTF { #2 }
8509   {
8510     \str_case:nVTF { #1 } \c_@@_integers_alist_tl
8511     {
8512       \flag_raise:n { nicematrix }
8513       \int_if_even:nTF { \flag_height:n { nicematrix } }
8514       { \int_eval:n { #1 + \l_@@_first_i_tl - 1 } }
8515       { \int_eval:n { #1 + \l_@@_first_j_tl - 1 } }
8516     }
8517     { #1 }
8518   }

```

If there is an hyphen, we have to see whether we have a node of the form $i-j$, $\text{row-}i$ or $\text{col-}j$.

```

8519     { \@@_pgfpointanchor_iii:w { #1 } #2 }
8520   }

```

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`).

```

8521 \cs_new:Npn \@@_pgfpointanchor_iii:w #1 #2 -
8522 {
8523   \str_case:nnF { #1 }
8524   {
8525     { row } { row - \int_eval:n { #2 + \l_@@_first_i_tl - 1 } }
8526     { col } { col - \int_eval:n { #2 + \l_@@_first_j_tl - 1 } }
8527   }

```

Now the case of a node of the form $i-j$.

```

8528   {
8529     \int_eval:n { #1 + \l_@@_first_i_tl - 1 }
8530     - \int_eval:n { #2 + \l_@@_first_j_tl - 1 }
8531   }
8532 }

```

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`).

```

8533 \cs_new_protected:Npn \@@_node_left:nn #1 #2
8534 {
8535   \pgfnode
8536   { rectangle }

```

```

8537 { east }
8538 {
8539   \nullfont
8540   \c_math_toggle_token
8541   \@@_color:o \l_@@_delimiters_color_tl
8542   \left #1
8543   \vcenter
8544   {
8545     \nullfont
8546     \hrule \@height \l_tmpa_dim
8547             \@depth \c_zero_dim
8548             \@width \c_zero_dim
8549   }
8550   \right .
8551   \c_math_toggle_token
8552 }
8553 { #2 }
8554 { }
8555 }

```

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.

```

8556 \cs_new_protected:Npn \@@_node_right:nnnn #1 #2 #3 #4
8557 {
8558   \pgfnode
8559   { rectangle }
8560   { west }
8561   {
8562     \nullfont
8563     \c_math_toggle_token
8564     \@@_color:o \l_@@_delimiters_color_tl
8565     \left .
8566     \vcenter
8567     {
8568       \nullfont
8569       \hrule \@height \l_tmpa_dim
8570               \@depth \c_zero_dim
8571               \@width \c_zero_dim
8572     }
8573     \right #1
8574     \tl_if_empty:nF { #3 } { _ { \smash { #3 } } }
8575     ^ { \smash { #4 } }
8576     \c_math_toggle_token
8577   }
8578   { #2 }
8579   { }
8580 }

```

35 Les commandes `\UnderBrace` et `\OverBrace`

The following commands will be linked to `\UnderBrace` and `\OverBrace` in the `\CodeAfter`.

```

8581 \NewDocumentCommand \@@_UnderBrace { 0 { } m m m 0 { } }
8582 {
8583   \peek_remove_spaces:n
8584   { \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { under } }
8585 }

```

```

8586 \NewDocumentCommand \@@_OverBrace { 0 { } m m m 0 { } }
8587 {
8588   \peek_remove_spaces:n
8589   { \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { over } }
8590 }

8591 \keys_define:nn { NiceMatrix / Brace }
8592 {
8593   left-shorten .bool_set:N = \l_@@_brace_left_shorten_bool ,
8594   left-shorten .default:n = true ,
8595   right-shorten .bool_set:N = \l_@@_brace_right_shorten_bool ,
8596   shorten .meta:n = { left-shorten , right-shorten } ,
8597   right-shorten .default:n = true ,
8598   yshift .dim_set:N = \l_@@_brace_yshift_dim ,
8599   yshift .value_required:n = true ,
8600   yshift .initial:n = \c_zero_dim ,
8601   color .tl_set:N = \l_tmpa_tl ,
8602   color .value_required:n = true ,
8603   unknown .code:n = \@@_error:n { Unknown~key~for~Brace }
8604 }

```

#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*.

```

8605 \cs_new_protected:Npn \@@_brace:nnnnn #1 #2 #3 #4 #5
8606 {
8607   \group_begin:

```

The four following token lists correspond to the position of the sub-matrix to which a brace will be attached.

```

8608   \@@_compute_i_j:nn { #1 } { #2 }
8609   \bool_lazy_or:nnTF
8610     { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
8611     { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
8612     {
8613       \str_if_eq:nnTF { #5 } { under }
8614       { \@@_error:nn { Construct-too-large } { \UnderBrace } }
8615       { \@@_error:nn { Construct-too-large } { \OverBrace } }
8616     }
8617   {
8618     \tl_clear:N \l_tmpa_tl
8619     \keys_set:nn { NiceMatrix / Brace } { #4 }
8620     \tl_if_empty:NF \l_tmpa_tl { \color { \l_tmpa_tl } }
8621     \pgfpicture
8622     \pgfrememberpicturepositiononpagetrue
8623     \pgf@relevantforpicturesizefalse
8624     \bool_if:NT \l_@@_brace_left_shorten_bool
8625     {
8626       \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
8627       \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
8628       {
8629         \cs_if_exist:cT
8630           { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
8631           {
8632             \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
8633             \dim_set:Nn \l_@@_x_initial_dim
8634               { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
8635           }
8636       }
8637     }
8638     \bool_lazy_or:nnT
8639       { \bool_not_p:n \l_@@_brace_left_shorten_bool }
8640       { \dim_compare_p:nNn \l_@@_x_initial_dim = \c_max_dim }
8641     {

```

```

8642     \@@_qpoint:n { col - \l_@@_first_j_tl }
8643     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
8644 }
8645 \bool_if:NT \l_@@_brace_right_shorten_bool
8646 {
8647     \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
8648     \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
8649     {
8650         \cs_if_exist:cT
8651         { \pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
8652         {
8653             \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
8654             \dim_set:Nn \l_@@_x_final_dim
8655             { \dim_max:nn \l_@@_x_final_dim \pgf@x }
8656         }
8657     }
8658 }
8659 \bool_lazy_or:nnT
8660 { \bool_not_p:n \l_@@_brace_right_shorten_bool }
8661 { \dim_compare_p:nNn \l_@@_x_final_dim = { - \c_max_dim } }
8662 {
8663     \@@_qpoint:n { col - \int_eval:n { \l_@@_last_j_tl + 1 } }
8664     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
8665 }
8666 \pgfset { inner~sep = \c_zero_dim }
8667 \str_if_eq:nnTF { #5 } { under }
8668 { \@@_underbrace_i:n { #3 } }
8669 { \@@_overbrace_i:n { #3 } }
8670 \endpgfpicture
8671 }
8672 \group_end:
8673 }

```

The argument is the text to put above the brace.

```

8674 \cs_new_protected:Npn \@@_overbrace_i:n #1
8675 {
8676     \@@_qpoint:n { row - \l_@@_first_i_tl }
8677     \pgftransformshift
8678     {
8679         \pgfpoint
8680         { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
8681         { \pgf@y + \l_@@_brace_yshift_dim - 3 pt }
8682     }
8683     \pgfnode
8684     { rectangle }
8685     { south }
8686     {
8687         \vtop
8688         {
8689             \group_begin:
8690             \everycr { }
8691             \halign
8692             {
8693                 \hfil ## \hfil \crcr
8694                 \@@_math_toggle: #1 \@@_math_toggle: \cr
8695                 \noalign { \skip_vertical:n { 3 pt } \nointerlineskip }
8696                 \c_math_toggle_token
8697                 \overbrace
8698                 {
8699                     \hbox_to_wd:nn
8700                     { \l_@@_x_final_dim - \l_@@_x_initial_dim }
8701                     { }
8702                 }
8703                 \c_math_toggle_token

```

```

8704         \cr
8705     }
8706     \group_end:
8707 }
8708 }
8709 { }
8710 { }
8711 }

```

The argument is the text to put under the brace.

```

8712 \cs_new_protected:Npn \@@_underbrace_i:n #1
8713 {
8714     \@@_qpoint:n { row - \int_eval:n { \l_@@_last_i_tl + 1 } }
8715     \pgftransformshift
8716     {
8717         \pgfpoint
8718         { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
8719         { \pgf@y - \l_@@_brace_yshift_dim + 3 pt }
8720     }
8721     \pgfnode
8722     { rectangle }
8723     { north }
8724     {
8725         \group_begin:
8726         \everycr { }
8727         \vbox
8728         {
8729             \halign
8730             {
8731                 \hfil ## \hfil \crrc
8732                 \c_math_toggle_token
8733                 \underbrace
8734                 {
8735                     \hbox_to_wd:nn
8736                     { \l_@@_x_final_dim - \l_@@_x_initial_dim }
8737                     { }
8738                 }
8739                 \c_math_toggle_token
8740                 \cr
8741                 \noalign { \skip_vertical:n { 3 pt } \nointerlineskip }
8742                 \@@_math_toggle: #1 \@@_math_toggle: \cr
8743             }
8744         }
8745         \group_end:
8746     }
8747     { }
8748     { }
8749 }

```

36 The command TikzEveryCell

```

8750 \bool_new:N \l_@@_not_empty_bool
8751 \bool_new:N \l_@@_empty_bool
8752
8753 \keys_define:nn { NiceMatrix / TikzEveryCell }
8754 {
8755     not-empty .code:n =
8756         \bool_lazy_or:nnTF
8757         \l_@@_in_code_after_bool

```

```

8758     \g_@@_recreate_cell_nodes_bool
8759     { \bool_set_true:N \l_@@_not_empty_bool }
8760     { \@@_error:n { detection~of~empty~cells } } ,
8761 not-empty .value_forbidden:n = true ,
8762 empty .code:n =
8763     \bool_lazy_or:nnTF
8764     \l_@@_in_code_after_bool
8765     \g_@@_recreate_cell_nodes_bool
8766     { \bool_set_true:N \l_@@_empty_bool }
8767     { \@@_error:n { detection~of~empty~cells } } ,
8768 empty .value_forbidden:n = true ,
8769 unknown .code:n = \@@_error:n { Unknown~key~for~TikzEveryCell }
8770 }
8771
8772
8773 \NewDocumentCommand { \@@_TikzEveryCell } { 0 { } m }
8774 {
8775     \IfPackageLoadedTF { tikz }
8776     {
8777         \group_begin:
8778         \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.

```

8779     \tl_set:Nn \l_tmpa_tl { { #2 } }
8780     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
8781     { \@@_for_a_block:nnnnn ##1 }
8782     \@@_all_the_cells:
8783     \group_end:
8784 }
8785 { \@@_error:n { TikzEveryCell~without~tikz } }
8786 }
8787
8788 \tl_new:N \@@_i_tl
8789 \tl_new:N \@@_j_tl
8790
8791 \cs_new_protected:Nn \@@_all_the_cells:
8792 {
8793     \int_step_variable:nNn { \int_use:c { c@iRow } } \@@_i_tl
8794     {
8795         \int_step_variable:nNn { \int_use:c { c@jCol } } \@@_j_tl
8796         {
8797             \cs_if_exist:cF { cell - \@@_i_tl - \@@_j_tl }
8798             {
8799                 \exp_args:NNe \seq_if_in:NnF \l_@@_corners_cells_seq
8800                 { \@@_i_tl - \@@_j_tl }
8801                 {
8802                     \bool_set_false:N \l_tmpa_bool
8803                     \cs_if_exist:cTF
8804                     { pgf @ sh @ ns @ \@@_env: - \@@_i_tl - \@@_j_tl }
8805                     {
8806                         \bool_if:NF \l_@@_empty_bool
8807                         { \bool_set_true:N \l_tmpa_bool }
8808                     }
8809                     {
8810                         \bool_if:NF \l_@@_not_empty_bool
8811                         { \bool_set_true:N \l_tmpa_bool }
8812                     }
8813                     \bool_if:NT \l_tmpa_bool
8814                     {
8815                         \@@_block_tikz:nnnnV
8816                         \@@_i_tl \@@_j_tl \@@_i_tl \@@_j_tl \l_tmpa_tl
8817                     }
8818                 }

```

```

8819         }
8820     }
8821 }
8822 }
8823
8824 \cs_new_protected:Nn \@@_for_a_block:nnnnn
8825 {
8826     \bool_if:NF \l_@@_empty_bool
8827     {
8828         \@@_block_tikz:nnnnV
8829         { #1 } { #2 } { #3 } { #4 } \l_tmpa_tl
8830     }
8831     \@@_mark_cells_of_block:nnnn { #1 } { #2 } { #3 } { #4 }
8832 }
8833
8834 \cs_new_protected:Nn \@@_mark_cells_of_block:nnnn
8835 {
8836     \int_step_inline:nnn { #1 } { #3 }
8837     {
8838         \int_step_inline:nnn { #2 } { #4 }
8839         { \cs_set:cpn { cell - ##1 - #####1 } { } }
8840     }
8841 }

```

37 The command \ShowCellNames

```

8842 \NewDocumentCommand \@@_ShowCellNames_CodeBefore { }
8843 {
8844     \dim_zero_new:N \g_@@_tmpc_dim
8845     \dim_zero_new:N \g_@@_tmpd_dim
8846     \dim_zero_new:N \g_@@_tmpe_dim
8847     \int_step_inline:nn \c@iRow
8848     {
8849         \begin { pgfpicture }
8850         \@@_qpoint:n { row - ##1 }
8851         \dim_set_eq:NN \l_tmpa_dim \pgf@y
8852         \@@_qpoint:n { row - \int_eval:n { ##1 + 1 } }
8853         \dim_gset:Nn \g_tmpa_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
8854         \dim_gset:Nn \g_tmpb_dim { \l_tmpa_dim - \pgf@y }
8855         \bool_if:NTF \l_@@_in_code_after_bool
8856         \end { pgfpicture }
8857         \int_step_inline:nn \c@jCol
8858         {
8859             \hbox_set:Nn \l_tmpa_box
8860             { \normalfont \Large \color { red ! 50 } ##1 - #####1 }
8861             \begin { pgfpicture }
8862             \@@_qpoint:n { col - #####1 }
8863             \dim_gset_eq:NN \g_@@_tmpc_dim \pgf@x
8864             \@@_qpoint:n { col - \int_eval:n { #####1 + 1 } }
8865             \dim_gset:Nn \g_@@_tmpd_dim { \pgf@x - \g_@@_tmpc_dim }
8866             \dim_gset_eq:NN \g_@@_tmpe_dim \pgf@x
8867             \endpgfpicture
8868             \end { pgfpicture }
8869             \fp_set:Nn \l_tmpa_fp
8870             {
8871                 \fp_min:nn
8872                 {
8873                     \fp_min:nn
8874                     {
8875                         \dim_ratio:nn
8876                         { \g_@@_tmpd_dim }
8877                         { \box_wd:N \l_tmpa_box }

```

```

8878         }
8879         {
8880             \dim_ratio:nn
8881             { \g_tmpb_dim }
8882             { \box_ht_plus_dp:N \l_tmpa_box }
8883         }
8884     }
8885     { 1.0 }
8886 }
8887 \box_scale:Nnn \l_tmpa_box
8888 { \fp_use:N \l_tmpa_fp }
8889 { \fp_use:N \l_tmpa_fp }
8890 \pgfpicture
8891 \pgfrememberpicturepositiononpagetrue
8892 \pgf@relevantforpicturesizefalse
8893 \pgftransformshift
8894 {
8895     \pgfpoint
8896     { 0.5 * ( \g_@@_tmpc_dim + \g_@@_tmpe_dim ) }
8897     { \dim_use:N \g_tmpa_dim }
8898 }
8899 \pgfnode
8900 { rectangle }
8901 { center }
8902 { \box_use:N \l_tmpa_box }
8903 { }
8904 { }
8905 \endpgfpicture
8906 }
8907 }
8908 }
8909 \NewDocumentCommand \@@_ShowCellNames { }
8910 {
8911     \bool_if:NT \l_@@_in_code_after_bool
8912     {
8913         \pgfpicture
8914         \pgfrememberpicturepositiononpagetrue
8915         \pgf@relevantforpicturesizefalse
8916         \pgfpathrectanglecorners
8917         { \@@_qpoint:n { 1 } }
8918         {
8919             \@@_qpoint:n
8920             { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
8921         }
8922         \pgfsetfillopacity { 0.75 }
8923         \pgfsetfillcolor { white }
8924         \pgfusepathqfill
8925         \endpgfpicture
8926     }
8927     \dim_zero_new:N \g_@@_tmpc_dim
8928     \dim_zero_new:N \g_@@_tmpd_dim
8929     \dim_zero_new:N \g_@@_tmpe_dim
8930     \int_step_inline:nn \c@iRow
8931     {
8932         \bool_if:NTF \l_@@_in_code_after_bool
8933         {
8934             \pgfpicture
8935             \pgfrememberpicturepositiononpagetrue
8936             \pgf@relevantforpicturesizefalse
8937         }
8938         { \begin { pgfpicture } }
8939         \@@_qpoint:n { row - ##1 }
8940         \dim_set_eq:NN \l_tmpa_dim \pgf@y

```

```

8941 \@@_qpoint:n { row - \int_eval:n { ##1 + 1 } }
8942 \dim_gset:Nn \g_tmpa_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
8943 \dim_gset:Nn \g_tmpb_dim { \l_tmpa_dim - \pgf@y }
8944 \bool_if:NTF \l_@@_in_code_after_bool
8945 { \endpgfpicture }
8946 { \end { pgfpicture } }
8947 \int_step_inline:nn \c@jCol
8948 {
8949   \hbox_set:Nn \l_tmpa_box
8950   {
8951     \normalfont \Large \sffamily \bfseries
8952     \bool_if:NTF \l_@@_in_code_after_bool
8953     { \color { red } }
8954     { \color { red ! 50 } }
8955     ##1 - ####1
8956   }
8957   \bool_if:NTF \l_@@_in_code_after_bool
8958   {
8959     \pgfpicture
8960     \pgfrememberpicturepositiononpagetrue
8961     \pgf@relevantforpicturesizefalse
8962   }
8963   { \begin { pgfpicture } }
8964   \@@_qpoint:n { col - ####1 }
8965   \dim_gset_eq:NN \g_@@_tmpc_dim \pgf@x
8966   \@@_qpoint:n { col - \int_eval:n { ####1 + 1 } }
8967   \dim_gset:Nn \g_@@_tmpd_dim { \pgf@x - \g_@@_tmpc_dim }
8968   \dim_gset_eq:NN \g_@@_tmpe_dim \pgf@x
8969   \bool_if:NTF \l_@@_in_code_after_bool
8970   { \endpgfpicture }
8971   { \end { pgfpicture } }
8972   \fp_set:Nn \l_tmpa_fp
8973   {
8974     \fp_min:nn
8975     {
8976       \fp_min:nn
8977       { \dim_ratio:nn \g_@@_tmpd_dim { \box_wd:N \l_tmpa_box } }
8978       { \dim_ratio:nn \g_tmpb_dim { \box_ht_plus_dp:N \l_tmpa_box } }
8979     }
8980     { 1.0 }
8981   }
8982   \box_scale:Nnn \l_tmpa_box { \fp_use:N \l_tmpa_fp } { \fp_use:N \l_tmpa_fp }
8983   \pgfpicture
8984   \pgfrememberpicturepositiononpagetrue
8985   \pgf@relevantforpicturesizefalse
8986   \pgftransformshift
8987   {
8988     \pgfpoint
8989     { 0.5 * ( \g_@@_tmpc_dim + \g_@@_tmpe_dim ) }
8990     { \dim_use:N \g_tmpa_dim }
8991   }
8992   \pgfnode
8993   { rectangle }
8994   { center }
8995   { \box_use:N \l_tmpa_box }
8996   { }
8997   { }
8998   \endpgfpicture
8999 }
9000 }
9001 }

```

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.

```
9002 \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.

```
9003 \bool_new:N \g_@@_footnote_bool
9004 \msg_new:nnnn { nicematrix } { Unknown~key~for~package }
9005 {
9006   The~key~'\l_keys_key_str'~is~unknown. \\
9007   That~key~will~be~ignored. \\
9008   For~a~list~of~the~available~keys,~type~H~<return>.
9009 }
9010 {
9011   The~available~keys~are~(in~alphabetic~order):~
9012   footnote,~
9013   footnotehyper,~
9014   messages-for-Overleaf,~
9015   no-test-for-array,~
9016   renew-dots,~and~
9017   renew-matrix.
9018 }
9019 \keys_define:nn { NiceMatrix / Package }
9020 {
9021   renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
9022   renew-dots .value_forbidden:n = true ,
9023   renew-matrix .code:n = \@@_renew_matrix: ,
9024   renew-matrix .value_forbidden:n = true ,
9025   messages-for-Overleaf .bool_set:N = \g_@@_messages_for_Overleaf_bool ,
9026   footnote .bool_set:N = \g_@@_footnote_bool ,
9027   footnotehyper .bool_set:N = \g_@@_footnotehyper_bool ,
9028   no-test-for-array .bool_set:N = \g_@@_no_test_for_array_bool ,
9029   no-test-for-array .default:n = true ,
9030   unknown .code:n = \@@_error:n { Unknown~key~for~package }
9031 }
9032 \ProcessKeysOptions { NiceMatrix / Package }

9033 \@@_msg_new:nn { footnote~with~footnotehyper~package }
9034 {
9035   You~can't~use~the~option~'footnote'~because~the~package~
9036   footnotehyper~has~already~been~loaded.~
9037   If~you~want,~you~can~use~the~option~'footnotehyper'~and~the~footnotes~
9038   within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
9039   of~the~package~footnotehyper.\\
9040   The~package~footnote~won't~be~loaded.
9041 }
9042 \@@_msg_new:nn { footnotehyper~with~footnote~package }
9043 {
9044   You~can't~use~the~option~'footnotehyper'~because~the~package~
9045   footnote~has~already~been~loaded.~
9046   If~you~want,~you~can~use~the~option~'footnote'~and~the~footnotes~
9047   within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
9048   of~the~package~footnote.\\
```

```

9049   The~package~footnotehyper~won't~be~loaded.
9050 }

```

```

9051 \bool_if:NT \g_@@_footnote_bool
9052 {

```

The class `beamer` has its own system to extract footnotes and that's why we have nothing to do if `beamer` is used.

```

9053   \IfClassLoadedTF { beamer }
9054   { \bool_set_false:N \g_@@_footnote_bool }
9055   {
9056     \IfPackageLoadedTF { footnotehyper }
9057     { \@@_error:n { footnote~with~footnotehyper~package } }
9058     { \usepackage { footnote } }
9059   }
9060 }
9061 \bool_if:NT \g_@@_footnotehyper_bool
9062 {

```

The class `beamer` has its own system to extract footnotes and that's why we have nothing to do if `beamer` is used.

```

9063   \IfClassLoadedTF { beamer }
9064   { \bool_set_false:N \g_@@_footnote_bool }
9065   {
9066     \IfPackageLoadedTF { footnote }
9067     { \@@_error:n { footnotehyper~with~footnote~package } }
9068     { \usepackage { footnotehyper } }
9069   }
9070   \bool_set_true:N \g_@@_footnote_bool
9071 }

```

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.

```

9072 \bool_new:N \l_@@_underscore_loaded_bool
9073 \IfPackageLoadedTF { underscore }
9074 { \bool_set_true:N \l_@@_underscore_loaded_bool }
9075 { }
9076 \hook_gput_code:nnn { begindocument } { . }
9077 {
9078   \bool_if:NF \l_@@_underscore_loaded_bool
9079   {
9080     \IfPackageLoadedTF { underscore }
9081     { \@@_error:n { underscore~after~nicematrix } }
9082     { }
9083   }
9084 }

```

40 Error messages of the package

```

9085 \bool_if:NTF \g_@@_messages_for_Overleaf_bool
9086 { \str_const:Nn \c_@@_available_keys_str { } }
9087 {
9088   \str_const:Nn \c_@@_available_keys_str
9089     { For~a~list~of~the~available~keys,~type~H~<return>. }
9090 }

9091 \seq_new:N \g_@@_types_of_matrix_seq
9092 \seq_gset_from_clist:Nn \g_@@_types_of_matrix_seq
9093 {
9094   NiceMatrix ,
9095   pNiceMatrix , bNiceMatrix , vNiceMatrix, BNiceMatrix, VNiceMatrix
9096 }
9097 \seq_gset_map_x:NNn \g_@@_types_of_matrix_seq \g_@@_types_of_matrix_seq
9098 { \tl_to_str:n { #1 } }

```

If the user uses too much columns, the command `\@@_error_too_much_cols:` is triggered. This command raises an error but also tries to give the best information to the user in the error message. The command `\seq_if_in:NoTF` is not expandable and that's why we can't put it in the error message itself. We have to do the test before the `\@@_fatal:n`.

```

9099 \cs_new_protected:Npn \@@_error_too_much_cols:
9100 {
9101   \seq_if_in:NoTF \g_@@_types_of_matrix_seq \g_@@_name_env_str
9102   {
9103     \int_compare:nNnTF \l_@@_last_col_int = { -2 }
9104     { \@@_fatal:n { too~much~cols~for~matrix } }
9105     {
9106       \int_compare:nNnTF \l_@@_last_col_int = { -1 }
9107       { \@@_fatal:n { too~much~cols~for~matrix } }
9108       {
9109         \bool_if:NF \l_@@_last_col_without_value_bool
9110         { \@@_fatal:n { too~much~cols~for~matrix~with~last~col } }
9111       }
9112     }
9113   }
9114   { \@@_fatal:nn { too~much~cols~for~array } }
9115 }

```

The following command must *not* be protected since it's used in an error message.

```

9116 \cs_new:Npn \@@_message_hdotsfor:
9117 {
9118   \tl_if_empty:oF \g_@@_HVDotsfor_lines_tl
9119   { ~Maybe~your~use~of~\token_to_str:N \Hdotsfor\ is~incorrect.}
9120 }

9121 \@@_msg_new:nn { hvlines,~rounded-corners~and~corners }
9122 {
9123   Incompatible~options.\\
9124   You~should~not~use~'hvlines',~'rounded-corners'~and~'corners'~at~this~time.\\
9125   The~output~will~not~be~reliable.
9126 }

9127 \@@_msg_new:nn { negative~weight }
9128 {
9129   Negative~weight.\\
9130   The~weight~of~the~'X'~columns~must~be~positive~and~you~have~used~
9131   the~value~'\int_use:N \l_@@_weight_int'.\\
9132   The~absolute~value~will~be~used.
9133 }

9134 \@@_msg_new:nn { last~col~not~used }
9135 {
9136   Column~not~used.\\

```

```

9137     The-key~'last-col'~is-in-force-but~you-have-not-used-that-last-column~
9138     in-your~\@@_full_name_env:~.~However,~you~can~go-on.
9139 }

9140 \@@_msg_new:nn { too-much-cols-for-matrix-with-last-col }
9141 {
9142     Too-much-columns.\\
9143     In-the-row~\int_eval:n { \c@iRow },~
9144     you-try-to-use-more-columns~
9145     than-allowed-by-your~\@@_full_name_env:~\@@_message_hdotsfor:~\
9146     The-maximal-number-of-columns-is~\int_eval:n { \l_@@_last_col_int - 1 }~
9147     (plus-the-exterior-columns).~This-error-is-fatal.
9148 }

9149 \@@_msg_new:nn { too-much-cols-for-matrix }
9150 {
9151     Too-much-columns.\\
9152     In-the-row~\int_eval:n { \c@iRow },~
9153     you-try-to-use-more-columns-than-allowed-by-your~
9154     \@@_full_name_env:~\@@_message_hdotsfor:~\ Recall~that~the~maximal~
9155     number~of~columns~for~a~matrix~(excepted~the~potential~exterior~
9156     columns)~is~fixed~by~the~LaTeX~counter~'MaxMatrixCols'.~
9157     Its~current~value~is~\int_use:N \c@MaxMatrixCols~(use~
9158     \token_to_str:N \setcounter~to~change~that~value).~
9159     This-error-is-fatal.
9160 }

9161 \@@_msg_new:nn { too-much-cols-for-array }
9162 {
9163     Too-much-columns.\\
9164     In-the-row~\int_eval:n { \c@iRow },~
9165     ~you-try-to-use-more-columns-than-allowed-by-your~
9166     \@@_full_name_env:~\@@_message_hdotsfor:~\ The~maximal~number-of~columns-is~
9167     \int_use:N \g_@@_static_num_of_col_int~
9168     ~(plus~the~potential~exterior~ones).
9169     This-error-is-fatal.
9170 }

9171 \@@_msg_new:nn { columns-not-used }
9172 {
9173     Columns-not-used.\\
9174     The-preamble-of~your~\@@_full_name_env:~\ announces~\int_use:N
9175     \g_@@_static_num_of_col_int~ columns-but-you-use-only~\int_use:N \c@jCol.\\
9176     The-columns-you-did-not-used-won't-be-created.\\
9177     You-won't-have-similar-error-till-the-end-of-the-document.
9178 }

9179 \@@_msg_new:nn { in-first-col }
9180 {
9181     Erroneous-use.\\
9182     You-can't-use-the-command~#1 in-the-first-column~(number~0)~of~the~array.\\
9183     That-command-will-be-ignored.
9184 }

9185 \@@_msg_new:nn { in-last-col }
9186 {
9187     Erroneous-use.\\
9188     You-can't-use-the-command~#1 in-the-last-column~(exterior)~of~the~array.\\
9189     That-command-will-be-ignored.
9190 }

9191 \@@_msg_new:nn { in-first-row }
9192 {
9193     Erroneous-use.\\
9194     You-can't-use-the-command~#1 in-the-first-row~(number~0)~of~the~array.\\
9195     That-command-will-be-ignored.
9196 }

```

```

9197 \@@_msg_new:nn { in~last~row }
9198 {
9199   You~can't~use~the~command~\#1~in~the~last~row~(exterior)~of~the~array.\\
9200   That~command~will~be~ignored.
9201 }
9202 \@@_msg_new:nn { caption~outside~float }
9203 {
9204   Key~caption~forbidden.\\
9205   You~can't~use~the~key~'caption'~because~you~are~not~in~a~floating~
9206   environment.~This~key~will~be~ignored.
9207 }
9208 \@@_msg_new:nn { short~caption~without~caption }
9209 {
9210   You~should~not~use~the~key~'short~caption'~without~'caption'.~
9211   However,~your~'short~caption'~will~be~used~as~'caption'.
9212 }
9213 \@@_msg_new:nn { double~closing~delimiter }
9214 {
9215   Double~delimiter.\\
9216   You~can't~put~a~second~closing~delimiter~"#1"~just~after~a~first~closing~
9217   delimiter.~This~delimiter~will~be~ignored.
9218 }
9219 \@@_msg_new:nn { delimiter~after~opening }
9220 {
9221   Double~delimiter.\\
9222   You~can't~put~a~second~delimiter~"#1"~just~after~a~first~opening~
9223   delimiter.~That~delimiter~will~be~ignored.
9224 }
9225 \@@_msg_new:nn { bad~option~for~line~style }
9226 {
9227   Bad~line~style.\\
9228   Since~you~haven't~loaded~Tikz,~the~only~value~you~can~give~to~'line~style'~
9229   is~'standard'.~That~key~will~be~ignored.
9230 }
9231 \@@_msg_new:nn { Identical~notes~in~caption }
9232 {
9233   Identical~tabular~notes.\\
9234   You~can't~put~several~notes~with~the~same~content~in~
9235   \token_to_str:N \caption\ (but~you~can~in~the~main~tabular).\\
9236   If~you~go~on,~the~output~will~probably~be~erroneous.
9237 }
9238 \@@_msg_new:nn { tabularnote~below~the~tabular }
9239 {
9240   \token_to_str:N \tabularnote\ forbidden\\
9241   You~can't~use~\token_to_str:N \tabularnote\ in~the~caption~
9242   of~your~tabular~because~the~caption~will~be~composed~below~
9243   the~tabular.~If~you~want~the~caption~above~the~tabular~use~the~
9244   key~'caption~above'~in~\token_to_str:N \NiceMatrixOptions.\\
9245   Your~\token_to_str:N \tabularnote\ will~be~discarded~and~
9246   no~similar~error~will~raised~in~this~document.
9247 }
9248 \@@_msg_new:nn { Unknown~key~for~rules }
9249 {
9250   Unknown~key.\\
9251   There~is~only~two~keys~available~here:~width~and~color.\\
9252   Your~key~'\l_keys_key_str'~will~be~ignored.
9253 }
9254 \@@_msg_new:nn { Unknown~key~for~TikzEveryCell }
9255 {
9256   Unknown~key.\\

```

```

9257     There~is~only~two~keys~available~here:~
9258     'empty'~and~'not-empty'.\\
9259     Your~key~'\l_keys_key_str'~will~be~ignored.
9260 }

9261 \@@_msg_new:nn { Unknown~key~for~rotate }
9262 {
9263     Unknown~key.\\
9264     The~only~key~available~here~is~'c'.\\
9265     Your~key~'\l_keys_key_str'~will~be~ignored.
9266 }

9267 \@@_msg_new:nnn { Unknown~key~for~custom~line }
9268 {
9269     Unknown~key.\\
9270     The~key~'\l_keys_key_str'~is~unknown~in~a~'custom~line'.~
9271     It~you~go~on,~you~will~probably~have~other~errors. \\
9272     \c_@@_available_keys_str
9273 }
9274 {
9275     The~available~keys~are~(in~alphabetic~order):~
9276     ccommand,~
9277     color,~
9278     command,~
9279     dotted,~
9280     letter,~
9281     multiplicity,~
9282     sep-color,~
9283     tikz,~and~total-width.
9284 }

9285 \@@_msg_new:nnn { Unknown~key~for~xdots }
9286 {
9287     Unknown~key.\\
9288     The~key~'\l_keys_key_str'~is~unknown~for~a~command~for~drawing~dotted~rules.\\
9289     \c_@@_available_keys_str
9290 }
9291 {
9292     The~available~keys~are~(in~alphabetic~order):~
9293     'color',~
9294     'horizontal-labels',~
9295     'inter',~
9296     'line-style',~
9297     'radius',~
9298     'shorten',~
9299     'shorten-end'~and~'shorten-start'.
9300 }

9301 \@@_msg_new:nn { Unknown~key~for~rowcolors }
9302 {
9303     Unknown~key.\\
9304     As~for~now,~there~is~only~two~keys~available~here:~'cols'~and~'respect-blocks'~
9305     (and~you~try~to~use~'\l_keys_key_str')\\
9306     That~key~will~be~ignored.
9307 }

9308 \@@_msg_new:nn { label~without~caption }
9309 {
9310     You~can't~use~the~key~'label'~in~your~'{NiceTabular}'~because~
9311     you~have~not~used~the~key~'caption'.~The~key~'label'~will~be~ignored.
9312 }

9313 \@@_msg_new:nn { W~warning }
9314 {
9315     Line~\msg_line_number:~The~cell~is~too~wide~for~your~column~'W'~
9316     (row~\int_use:N \c@iRow).
9317 }

```

```

9318 \@@_msg_new:nn { Construct-too-large }
9319 {
9320   Construct-too-large.\
9321   Your~command~\token_to_str:N #1
9322   can't~be~drawn~because~your~matrix~is~too~small.\
9323   That~command~will~be~ignored.
9324 }
9325 \@@_msg_new:nn { underscore-after-nicematrix }
9326 {
9327   Problem-with~'underscore'.\
9328   The~package~'underscore'~should~be~loaded~before~'nicematrix'.~
9329   You~can~go~on~but~you~won't~be~able~to~write~something~such~as:\
9330   '\token_to_str:N \Cdots\token_to_str:N _{n~\token_to_str:N \text{~times}}'.
9331 }
9332 \@@_msg_new:nn { ampersand-in-light-syntax }
9333 {
9334   Ampersand~forbidden.\
9335   You~can't~use~an~ampersand~(\token_to_str:N &)~to~separate~columns~because~
9336   ~the~key~'light-syntax'~is~in~force.~This~error~is~fatal.
9337 }
9338 \@@_msg_new:nn { double-backslash-in-light-syntax }
9339 {
9340   Double~backslash~forbidden.\
9341   You~can't~use~\token_to_str:N
9342   \~to~separate~rows~because~the~key~'light-syntax'~
9343   is~in~force.~You~must~use~the~character~'\l_@@_end_of_row_tl'~
9344   (set~by~the~key~'end-of-row').~This~error~is~fatal.
9345 }
9346 \@@_msg_new:nn { hlines-with-color }
9347 {
9348   Incompatible~keys.\
9349   You~can't~use~the~keys~'hlines',~'vlines'~or~'hvlines'~for~a~
9350   '\token_to_str:N \Block'~when~the~key~'color'~or~'draw'~is~used.\
9351   Maybe~it~will~possible~in~future~version.\
9352   Your~key~will~be~discarded.
9353 }
9354 \@@_msg_new:nn { bad-value-for-baseline }
9355 {
9356   Bad~value~for~baseline.\
9357   The~value~given~to~'baseline'~(\int_use:N \l_tmpa_int)~is~not~
9358   valid.~The~value~must~be~between~\int_use:N \l_@@_first_row_int\ and~
9359   \int_use:N \g_@@_row_total_int\ or~equal~to~'t',~'c'~or~'b'~or~of~
9360   the~form~'line-i'~.\
9361   A~value~of~1~will~be~used.
9362 }
9363 \@@_msg_new:nn { detection-of-empty-cells }
9364 {
9365   Problem-with~'not-empty'\
9366   For~technical~reasons,~you~must~activate~
9367   'create-cell-nodes'~in~\token_to_str:N \CodeBefore\
9368   in~order~to~use~the~key~'\l_keys_key_str'~.\
9369   That~key~will~be~ignored.
9370 }
9371 \@@_msg_new:nn { siunitx-not-loaded }
9372 {
9373   siunitx~not~loaded\
9374   You~can't~use~the~columns~'S'~because~'siunitx'~is~not~loaded.\
9375   That~error~is~fatal.
9376 }
9377 \@@_msg_new:nn { ragged2e-not-loaded }

```

```

9378 {
9379     You-have-to-load~'ragged2e'~in-order-to-use-the-key~'\l_keys_key_str'~in~
9380     your~column~'\l_@@_vpos_col_str'~(or~'X')~.~The-key~'\str_lowercase:V
9381     \l_keys_key_str'~will-be-used-instead.
9382 }
9383 \@@_msg_new:nn { Invalid-name }
9384 {
9385     Invalid-name.\\
9386     You-can't-give-the-name~'\l_keys_value_tl'~to-a~\token_to_str:N
9387     \SubMatrix\ of~your~\@@_full_name_env:~.\\
9388     A-name-must-be-accepted-by-the-regular-expression~[A-Za-z][A-Za-z0-9]*.\\
9389     This-key-will-be-ignored.
9390 }
9391 \@@_msg_new:nn { Wrong-line-in-SubMatrix }
9392 {
9393     Wrong-line.\\
9394     You-try-to-draw-a~#1~line-of-number~'#2'~in-a~
9395     \token_to_str:N \SubMatrix\ of~your~\@@_full_name_env:\ but-that~
9396     number-is-not-valid.~It-will-be-ignored.
9397 }
9398 \@@_msg_new:nn { Impossible-delimiter }
9399 {
9400     Impossible-delimiter.\\
9401     It's-impossible-to-draw-the~#1~delimiter-of~your~
9402     \token_to_str:N \SubMatrix\ because-all-the-cells-are-empty~
9403     in-that-column.
9404     \bool_if:NT \l_@@_submatrix_slim_bool
9405     { ~Maybe-you-should-try-without-the-key~'slim'. } \\
9406     This~\token_to_str:N \SubMatrix\ will-be-ignored.
9407 }
9408 \@@_msg_new:nnn { width-without-X-columns }
9409 {
9410     You-have-used-the-key~'width'~but-you-have-put-no~'X'~column.~
9411     That-key-will-be-ignored.
9412 }
9413 {
9414     This-message-is-the-message~'width-without-X-columns'~
9415     of-the-module~'nicematrix'.~
9416     The-experimented-users-can-disable-that-message-with~
9417     \token_to_str:N \msg_redirect_name:nnn.\\
9418 }
9419
9420 \@@_msg_new:nn { key-multiplicity-with-dotted }
9421 {
9422     Incompatible-keys. \\
9423     You-have-used-the-key~'multiplicity'~with-the-key~'dotted'~
9424     in-a~'custom-line'.~They-are-incompatible. \\
9425     The-key~'multiplicity'~will-be-discarded.
9426 }
9427 \@@_msg_new:nn { empty-environment }
9428 {
9429     Empty-environment.\\
9430     Your~\@@_full_name_env:\ is-empty.~This-error-is-fatal.
9431 }
9432 \@@_msg_new:nn { No-letter-and-no-command }
9433 {
9434     Erroneous-use.\\
9435     Your-use-of~'custom-line'~is-no-op~since-you-don't-have-used-the~
9436     key~'letter'~(for-a-letter-for-vertical-rules)~nor-the-keys~'command'~or~
9437     ~'ccommand'~(to-draw-horizontal-rules).\\
9438     However,~you-can-go-on.

```

```

9439 }
9440 \@@_msg_new:nn { Forbidden-letter }
9441 {
9442   Forbidden-letter.\
9443   You-can't-use-the-letter~'#1'~for-a-customized-line.\
9444   It-will-be-ignored.
9445 }
9446 \@@_msg_new:nn { Several-letters }
9447 {
9448   Wrong-name.\
9449   You-must-use-only-one-letter-as-value-for-the-key~'letter'~(and-you-
9450   have-used~'\l_@@_letter_str').\
9451   It-will-be-ignored.
9452 }
9453 \@@_msg_new:nn { Delimiter-with-small }
9454 {
9455   Delimiter~forbidden.\
9456   You-can't-put-a-delimiter-in-the-preamble-of-your~\@@_full_name_env:\
9457   because-the-key~'small'~is-in-force.\
9458   This-error-is-fatal.
9459 }
9460 \@@_msg_new:nn { unknown-cell-for-line-in-CodeAfter }
9461 {
9462   Unknown-cell.\
9463   Your-command~\token_to_str:N\line\{#1\}\{#2\}~in-
9464   the~\token_to_str:N \CodeAfter\ of-your~\@@_full_name_env:\
9465   can't-be-executed-because-a-cell-doesn't-exist.\
9466   This-command~\token_to_str:N \line\ will-be-ignored.
9467 }
9468 \@@_msg_new:nnn { Duplicate-name-for-SubMatrix }
9469 {
9470   Duplicate-name.\
9471   The-name~'#1'~is-already-used-for-a~\token_to_str:N \SubMatrix\
9472   in-this~\@@_full_name_env:.\
9473   This-key-will-be-ignored.\
9474   \bool_if:NF \g_@@_messages_for_Overleaf_bool
9475   { For-a-list-of-the-names-already-used,~type-H<return>. }
9476 }
9477 {
9478   The-names-already-defined-in-this~\@@_full_name_env:\ are:~
9479   \seq_use:Nnnn \g_@@_submatrix_names_seq { ~and~ } { ,~ } { ~and~ }.
9480 }
9481 \@@_msg_new:nn { r-or-l-with-preamble }
9482 {
9483   Erroneous-use.\
9484   You-can't-use-the-key~'\l_keys_key_str'~in-your~\@@_full_name_env:~
9485   You-must-specify-the-alignment-of-your-columns-with-the-preamble-of-
9486   your~\@@_full_name_env:.\
9487   This-key-will-be-ignored.
9488 }
9489 \@@_msg_new:nn { Hdotsfor-in-col-0 }
9490 {
9491   Erroneous-use.\
9492   You-can't-use~\token_to_str:N \Hdotsfor\ in-an-exterior-column-of-
9493   the-array.~This-error-is-fatal.
9494 }
9495 \@@_msg_new:nn { bad-corner }
9496 {
9497   Bad-corner.\
9498   #1-is-an-incorrect-specification-for-a-corner~(in-the-key~

```

```

9499     'corners').~The~available~values~are:~NW,~SW,~NE~and~SE.\\
9500     This~specification~of~corner~will~be~ignored.
9501 }

9502 \@@_msg_new:nn { bad~border }
9503 {
9504     Bad~border.\\
9505     \l_keys_key_str\space~is~an~incorrect~specification~for~a~border~
9506     (in~the~key~'borders'~of~the~command~\token_to_str:N \Block).~
9507     The~available~values~are:~left,~right,~top~and~bottom~(and~you~can~
9508     also~use~the~key~'tikz'
9509     \IfPackageLoadedTF { tikz }
9510         { }
9511         {~if~you~load~the~LaTeX~package~'tikz'}).\\
9512     This~specification~of~border~will~be~ignored.
9513 }

9514 \@@_msg_new:nn { TikzEveryCell~without~tikz }
9515 {
9516     TikZ~not~loaded.\\
9517     You~can't~use~\token_to_str:N \TikzEveryCell\
9518     because~you~have~not~loaded~tikz.~
9519     This~command~will~be~ignored.
9520 }

9521 \@@_msg_new:nn { tikz~key~without~tikz }
9522 {
9523     TikZ~not~loaded.\\
9524     You~can't~use~the~key~'tikz'~for~the~command~'\token_to_str:N
9525     \Block'~because~you~have~not~loaded~tikz.~
9526     This~key~will~be~ignored.
9527 }

9528 \@@_msg_new:nn { last~col~non~empty~for~NiceArray }
9529 {
9530     Erroneous~use.\\
9531     In~the~\@@_full_name_env:,~you~must~use~the~key~
9532     'last~col'~without~value.\\
9533     However,~you~can~go~on~for~this~time~
9534     (the~value~'\l_keys_value_tl'~will~be~ignored).
9535 }

9536 \@@_msg_new:nn { last~col~non~empty~for~NiceMatrixOptions }
9537 {
9538     Erroneous~use.\\
9539     In~\token_to_str:N \NiceMatrixOptions,~you~must~use~the~key~
9540     'last~col'~without~value.\\
9541     However,~you~can~go~on~for~this~time~
9542     (the~value~'\l_keys_value_tl'~will~be~ignored).
9543 }

9544 \@@_msg_new:nn { Block~too~large~1 }
9545 {
9546     Block~too~large.\\
9547     You~try~to~draw~a~block~in~the~cell~#1-#2~of~your~matrix~but~the~matrix~is~
9548     too~small~for~that~block. \\
9549     This~block~and~maybe~others~will~be~ignored.
9550 }

9551 \@@_msg_new:nn { Block~too~large~2 }
9552 {
9553     Block~too~large.\\
9554     The~preamble~of~your~\@@_full_name_env:\ announces~\int_use:N
9555     \g_@@_static_num_of_col_int\
9556     columns~but~you~use~only~\int_use:N \c_jCol\ and~that's~why~a~block~
9557     specified~in~the~cell~#1-#2~can't~be~drawn.~You~should~add~some~ampersands~
9558     (&)~at~the~end~of~the~first~row~of~your~\@@_full_name_env:.\
9559     This~block~and~maybe~others~will~be~ignored.

```

```

9560 }
9561 \@@_msg_new:nn { unknown~column~type }
9562 {
9563   Bad~column~type.\\
9564   The~column~type~'#1'~in~your~\@@_full_name_env:\
9565   is~unknown. \\
9566   This~error~is~fatal.
9567 }
9568 \@@_msg_new:nn { unknown~column~type~S }
9569 {
9570   Bad~column~type.\\
9571   The~column~type~'S'~in~your~\@@_full_name_env:\ is~unknown. \\
9572   If~you~want~to~use~the~column~type~'S'~of~siunitx,~you~should~
9573   load~that~package. \\
9574   This~error~is~fatal.
9575 }
9576 \@@_msg_new:nn { tabularnote~forbidden }
9577 {
9578   Forbidden~command.\\
9579   You~can't~use~the~command~\token_to_str:N\tabularnote\
9580   ~here.~This~command~is~available~only~in~
9581   \{NiceTabular\},~\{NiceTabular*\}~and~\{NiceTabularX\}~or~in~
9582   the~argument~of~a~command~\token_to_str:N \caption\ included~
9583   in~an~environment~{table}. \\
9584   This~command~will~be~ignored.
9585 }
9586 \@@_msg_new:nn { borders~forbidden }
9587 {
9588   Forbidden~key.\\
9589   You~can't~use~the~key~'borders'~of~the~command~\token_to_str:N \Block\
9590   because~the~option~'rounded~corners'~
9591   is~in~force~with~a~non~zero~value.\\
9592   This~key~will~be~ignored.
9593 }
9594 \@@_msg_new:nn { bottomrule~without~booktabs }
9595 {
9596   booktabs~not~loaded.\\
9597   You~can't~use~the~key~'tabular/bottomrule'~because~you~haven't~
9598   loaded~'booktabs'.\\
9599   This~key~will~be~ignored.
9600 }
9601 \@@_msg_new:nn { enumitem~not~loaded }
9602 {
9603   enumitem~not~loaded.\\
9604   You~can't~use~the~command~\token_to_str:N\tabularnote\
9605   ~because~you~haven't~loaded~'enumitem'.\\
9606   All~the~commands~\token_to_str:N\tabularnote\ will~be~
9607   ignored~in~the~document.
9608 }
9609 \@@_msg_new:nn { tikz~without~tikz }
9610 {
9611   Tikz~not~loaded.\\
9612   You~can't~use~the~key~'tikz'~here~because~Tikz~is~not~
9613   loaded.~If~you~go~on,~that~key~will~be~ignored.
9614 }
9615 \@@_msg_new:nn { tikz~in~custom~line~without~tikz }
9616 {
9617   Tikz~not~loaded.\\
9618   You~have~used~the~key~'tikz'~in~the~definition~of~a~
9619   customized~line~(with~'custom~line')~but~tikz~is~not~loaded.~

```

```

9620     You-can-go-on-but-you-will-have-another-error-if-you-actually-
9621     use-that-custom-line.
9622 }
9623 \@@_msg_new:nn { tikz-in-borders-without-tikz }
9624 {
9625     Tikz-not-loaded.\\
9626     You-have-used-the-key~'tikz'~in-a-key~'borders'~(of-a~
9627     command~'\token_to_str:N\Block')~but-tikz-is-not-loaded.~
9628     That-key-will-be-ignored.
9629 }
9630 \@@_msg_new:nn { without-color-inside }
9631 {
9632     If-order-to-use~\token_to_str:N \cellcolor,~\token_to_str:N \rowcolor,~
9633     \token_to_str:N \rowcolors\ or~\token_to_str:N \rowlistcolors\
9634     outside~\token_to_str:N \CodeBefore,~you~
9635     should-have-used-the-key~'color-inside'~in-your~\@@_full_name_env:~\\
9636     You-can-go-on-but-you-may-need-more-compilations.
9637 }
9638 \@@_msg_new:nn { color-in-custom-line-with-tikz }
9639 {
9640     Erroneous-use.\\
9641     In-a~'custom-line',~you-have-used-both~'tikz'~and~'color',~
9642     which-is-forbidden~(you-should-use~'color'~inside-the-key~'tikz')~.~
9643     The-key~'color'~will-be-discarded.
9644 }
9645 \@@_msg_new:nn { Wrong-last-row }
9646 {
9647     Wrong-number.\\
9648     You-have-used~'last-row=~\int_use:N \l_@@_last_row_int'~but-your~
9649     \@@_full_name_env:\ seems-to-have~\int_use:N \c@iRow \ rows.~
9650     If-you-go-on,~the-value-of~\int_use:N \c@iRow \ will-be-used-for~
9651     last-row.~You-can-avoid-this-problem-by-using~'last-row'~
9652     without-value~(more-compilations-might-be-necessary).
9653 }
9654 \@@_msg_new:nn { Yet-in-env }
9655 {
9656     Nested-environments.\\
9657     Environments-of-nicematrix-can't-be-nested.\\
9658     This-error-is-fatal.
9659 }
9660 \@@_msg_new:nn { Outside-math-mode }
9661 {
9662     Outside-math-mode.\\
9663     The~\@@_full_name_env:\ can-be-used-only-in-math-mode~
9664     (and-not-in~\token_to_str:N \vcenter).\\
9665     This-error-is-fatal.
9666 }
9667 \@@_msg_new:nn { One-letter-allowed }
9668 {
9669     Bad-name.\\
9670     The-value-of-key~'\l_keys_key_str'~must-be-of-length-1.\\
9671     It-will-be-ignored.
9672 }
9673 \@@_msg_new:nn { TabularNote-in-CodeAfter }
9674 {
9675     Environment~{TabularNote}~forbidden.\\
9676     You-must-use~{TabularNote}~at-the-end-of-your~{NiceTabular}~
9677     but~*before*~the~\token_to_str:N \CodeAfter.\\
9678     This-environment~{TabularNote}~will-be-ignored.
9679 }

```

```

9680 \@@_msg_new:nn { varwidth~not~loaded }
9681 {
9682   varwidth~not~loaded.\\
9683   You~can't~use~the~column~type~'V'~because~'varwidth'~is~not~
9684   loaded.\\
9685   Your~column~will~behave~like~'p'.
9686 }
9687 \@@_msg_new:nnn { Unknown~key~for~RulesBis }
9688 {
9689   Unknown~key.\\
9690   Your~key~'\l_keys_key_str'~is~unknown~for~a~rule.\\
9691   \c_@@_available_keys_str
9692 }
9693 {
9694   The~available~keys~are~(in~alphabetic~order):~
9695   color,~
9696   dotted,~
9697   multiplicity,~
9698   sep~color,~
9699   tikz,~and~total~width.
9700 }
9701
9702 \@@_msg_new:nnn { Unknown~key~for~Block }
9703 {
9704   Unknown~key.\\
9705   The~key~'\l_keys_key_str'~is~unknown~for~the~command~\token_to_str:N
9706   \Block.\\ It~will~be~ignored. \\
9707   \c_@@_available_keys_str
9708 }
9709 {
9710   The~available~keys~are~(in~alphabetic~order):~b,~B,~borders,~c,~draw,~fill,~
9711   hlines,~hvlines,~l,~line~width,~name,~opacity,~rounded~corners,~r,~
9712   respect~arraystretch,~t,~T,~tikz,~transparent~and~vlines.
9713 }
9714 \@@_msg_new:nnn { Unknown~key~for~Brace }
9715 {
9716   Unknown~key.\\
9717   The~key~'\l_keys_key_str'~is~unknown~for~the~commands~\token_to_str:N
9718   \UnderBrace\ and~\token_to_str:N \OverBrace.\\
9719   It~will~be~ignored. \\
9720   \c_@@_available_keys_str
9721 }
9722 {
9723   The~available~keys~are~(in~alphabetic~order):~color,~left~shorten,~
9724   right~shorten,~shorten~(which~fixes~both~left~shorten~and~
9725   right~shorten)~and~yshift.
9726 }
9727 \@@_msg_new:nnn { Unknown~key~for~CodeAfter }
9728 {
9729   Unknown~key.\\
9730   The~key~'\l_keys_key_str'~is~unknown.\\
9731   It~will~be~ignored. \\
9732   \c_@@_available_keys_str
9733 }
9734 {
9735   The~available~keys~are~(in~alphabetic~order):~
9736   delimiters/color,~
9737   rules~(with~the~subkeys~'color'~and~'width'),~
9738   sub~matrix~(several~subkeys)~
9739   and~xdots~(several~subkeys).~
9740   The~latter~is~for~the~command~\token_to_str:N \line.
9741 }

```

```

9742 \@@_msg_new:nnn { Unknown~key~for~CodeBefore }
9743 {
9744   Unknown~key.\\
9745   The~key~'\l_keys_key_str'~is~unknown.\\
9746   It~will~be~ignored. \\
9747   \c_@@_available_keys_str
9748 }
9749 {
9750   The~available~keys~are~(in~alphabetic~order):~
9751   create~cell~nodes,~
9752   delimiters/color~and~
9753   sub~matrix~(several~subkeys).
9754 }
9755 \@@_msg_new:nnn { Unknown~key~for~SubMatrix }
9756 {
9757   Unknown~key.\\
9758   The~key~'\l_keys_key_str'~is~unknown.\\
9759   That~key~will~be~ignored. \\
9760   \c_@@_available_keys_str
9761 }
9762 {
9763   The~available~keys~are~(in~alphabetic~order):~
9764   'delimiters/color',~
9765   'extra~height',~
9766   'hlines',~
9767   'hvlines',~
9768   'left~xshift',~
9769   'name',~
9770   'right~xshift',~
9771   'rules'~(with~the~subkeys~'color'~and~'width'),~
9772   'slim',~
9773   'vlines'~and~'xshift'~(which~sets~both~'left~xshift'~
9774   and~'right~xshift').\\
9775 }
9776 \@@_msg_new:nnn { Unknown~key~for~notes }
9777 {
9778   Unknown~key.\\
9779   The~key~'\l_keys_key_str'~is~unknown.\\
9780   That~key~will~be~ignored. \\
9781   \c_@@_available_keys_str
9782 }
9783 {
9784   The~available~keys~are~(in~alphabetic~order):~
9785   bottomrule,~
9786   code~after,~
9787   code~before,~
9788   detect~duplicates,~
9789   enumitem~keys,~
9790   enumitem~keys~para,~
9791   para,~
9792   label~in~list,~
9793   label~in~tabular~and~
9794   style.
9795 }
9796 \@@_msg_new:nnn { Unknown~key~for~RowStyle }
9797 {
9798   Unknown~key.\\
9799   The~key~'\l_keys_key_str'~is~unknown~for~the~command~
9800   \token_to_str:N \RowStyle. \\
9801   That~key~will~be~ignored. \\
9802   \c_@@_available_keys_str
9803 }
9804 {

```

```

9805 The~available~keys~are~(in~alphabetic~order):~
9806 'bold',~
9807 'cell-space-top-limit',~
9808 'cell-space-bottom-limit',~
9809 'cell-space-limits',~
9810 'color',~
9811 'nb-rows'~and~
9812 'rowcolor'.
9813 }
9814 \@@_msg_new:nnn { Unknown~key~for~NiceMatrixOptions }
9815 {
9816   Unknown~key.\\
9817   The~key~'\l_keys_key_str'~is~unknown~for~the~command~
9818   \token_to_str:N \NiceMatrixOptions. \\
9819   That~key~will~be~ignored. \\
9820   \c_@@_available_keys_str
9821 }
9822 {
9823   The~available~keys~are~(in~alphabetic~order):~
9824   allow-duplicate-names,~
9825   caption-above,~
9826   cell-space-bottom-limit,~
9827   cell-space-limits,~
9828   cell-space-top-limit,~
9829   code-for-first-col,~
9830   code-for-first-row,~
9831   code-for-last-col,~
9832   code-for-last-row,~
9833   corners,~
9834   custom-key,~
9835   create-extra-nodes,~
9836   create-medium-nodes,~
9837   create-large-nodes,~
9838   delimiters~(several~subkeys),~
9839   end-of-row,~
9840   first-col,~
9841   first-row,~
9842   hlines,~
9843   hvlines,~
9844   hvlines-except-borders,~
9845   last-col,~
9846   last-row,~
9847   left-margin,~
9848   light-syntax,~
9849   light-syntax-expanded,~
9850   matrix/columns-type,~
9851   no-cell-nodes,~
9852   notes~(several~subkeys),~
9853   nullify-dots,~
9854   pgf-node-code,~
9855   renew-dots,~
9856   renew-matrix,~
9857   respect-arraystretch,~
9858   rounded-corners,~
9859   right-margin,~
9860   rules~(with~the~subkeys~'color'~and~'width'),~
9861   small,~
9862   sub-matrix~(several~subkeys),~
9863   vlines,~
9864   xdots~(several~subkeys).
9865 }

```

For '{NiceArray}', the set of keys is the same as for {NiceMatrix} excepted that there is no l and r.

```

9866 \@@_msg_new:nnn { Unknown~key~for~NiceArray }
9867 {
9868   Unknown~key.\\
9869   The~key~'\l_keys_key_str'~is~unknown~for~the~environment~
9870   \{NiceArray\}. \\
9871   That~key~will~be~ignored. \\
9872   \c_@@_available_keys_str
9873 }
9874 {
9875   The~available~keys~are~(in~alphabetic~order):~
9876   b,~
9877   baseline,~
9878   c,~
9879   cell-space-bottom-limit,~
9880   cell-space-limits,~
9881   cell-space-top-limit,~
9882   code-after,~
9883   code-for-first-col,~
9884   code-for-first-row,~
9885   code-for-last-col,~
9886   code-for-last-row,~
9887   color-inside,~
9888   columns-width,~
9889   corners,~
9890   create-extra-nodes,~
9891   create-medium-nodes,~
9892   create-large-nodes,~
9893   extra-left-margin,~
9894   extra-right-margin,~
9895   first-col,~
9896   first-row,~
9897   hlines,~
9898   hvlines,~
9899   hvlines-except-borders,~
9900   last-col,~
9901   last-row,~
9902   left-margin,~
9903   light-syntax,~
9904   light-syntax-expanded,~
9905   name,~
9906   no-cell-nodes,~
9907   nullify-dots,~
9908   pgf-node-code,~
9909   renew-dots,~
9910   respect-arraystretch,~
9911   right-margin,~
9912   rounded-corners,~
9913   rules~(with~the~subkeys~'color'~and~'width'),~
9914   small,~
9915   t,~
9916   vlimes,~
9917   xdots/color,~
9918   xdots/shorten-start,~
9919   xdots/shorten-end,~
9920   xdots/shorten-and~
9921   xdots/line-style.
9922 }

```

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).

```

9923 \@@_msg_new:nnn { Unknown~key~for~NiceMatrix }
9924 {
9925   Unknown~key.\\
9926   The~key~'\l_keys_key_str'~is~unknown~for~the~

```

```

9927 \@@_full_name_env:. \
9928 That~key~will~be~ignored. \
9929 \c_@@_available_keys_str
9930 }
9931 {
9932   The~available~keys~are~(in~alphabetic~order):~
9933   b,~
9934   baseline,~
9935   c,~
9936   cell-space-bottom-limit,~
9937   cell-space-limits,~
9938   cell-space-top-limit,~
9939   code-after,~
9940   code-for-first-col,~
9941   code-for-first-row,~
9942   code-for-last-col,~
9943   code-for-last-row,~
9944   color-inside,~
9945   columns-type,~
9946   columns-width,~
9947   corners,~
9948   create-extra-nodes,~
9949   create-medium-nodes,~
9950   create-large-nodes,~
9951   extra-left-margin,~
9952   extra-right-margin,~
9953   first-col,~
9954   first-row,~
9955   hlines,~
9956   hvlines,~
9957   hvlines-except-borders,~
9958   l,~
9959   last-col,~
9960   last-row,~
9961   left-margin,~
9962   light-syntax,~
9963   light-syntax-expanded,~
9964   name,~
9965   no-cell-nodes,~
9966   nullify-dots,~
9967   pgf-node-code,~
9968   r,~
9969   renew-dots,~
9970   respect-arraystretch,~
9971   right-margin,~
9972   rounded-corners,~
9973   rules~(with~the~subkeys~'color'~and~'width'),~
9974   small,~
9975   t,~
9976   vlines,~
9977   xdots/color,~
9978   xdots/shorten-start,~
9979   xdots/shorten-end,~
9980   xdots/shorten-and~
9981   xdots/line-style.
9982 }
9983 \@@_msg_new:nnn { Unknown~key~for~NiceTabular }
9984 {
9985   Unknown~key.\
9986   The~key~'\l_keys_key_str'~is~unknown~for~the~environment~
9987   \{NiceTabular\}. \
9988   That~key~will~be~ignored. \
9989   \c_@@_available_keys_str

```

```

9990 }
9991 {
9992   The-available-keys-are~(in~alphabetic~order):~
9993   b,~
9994   baseline,~
9995   c,~
9996   caption,~
9997   cell-space-bottom-limit,~
9998   cell-space-limits,~
9999   cell-space-top-limit,~
10000   code-after,~
10001   code-for-first-col,~
10002   code-for-first-row,~
10003   code-for-last-col,~
10004   code-for-last-row,~
10005   color-inside,~
10006   columns-width,~
10007   corners,~
10008   custom-line,~
10009   create-extra-nodes,~
10010   create-medium-nodes,~
10011   create-large-nodes,~
10012   extra-left-margin,~
10013   extra-right-margin,~
10014   first-col,~
10015   first-row,~
10016   hlines,~
10017   hvlines,~
10018   hvlines-except-borders,~
10019   label,~
10020   last-col,~
10021   last-row,~
10022   left-margin,~
10023   light-syntax,~
10024   light-syntax-expanded,~
10025   name,~
10026   no-cell-nodes,~
10027   notes~(several~subkeys),~
10028   nullify-dots,~
10029   pgf-node-code,~
10030   renew-dots,~
10031   respect-arraystretch,~
10032   right-margin,~
10033   rounded-corners,~
10034   rules~(with~the~subkeys~'color'~and~'width'),~
10035   short-caption,~
10036   t,~
10037   tabularnote,~
10038   vlines,~
10039   xdots/color,~
10040   xdots/shorten-start,~
10041   xdots/shorten-end,~
10042   xdots/shorten-and~
10043   xdots/line-style.
10044 }

10045 \@@_msg_new:nnn { Duplicate-name }
10046 {
10047   Duplicate-name.\
10048   The-name-'\l_keys_value_tl'-is-already-used-and-you-shouldn't-use~
10049   the-same-environment-name-twice.~You-can-go-on,~but,~
10050   maybe,~you-will-have-incorrect-results-especially~
10051   if-you-use-'columns-width=auto'.~If-you-don't-want-to-see-this~
10052   message-again,~use-the-key~'allow-duplicate-names'~in~

```

```

10053     '\token_to_str:N \NiceMatrixOptions'.\\
10054     \bool_if:NF \g_@@_messages_for_Overleaf_bool
10055     { For~a~list~of~the~names~already~used,~type~H~<return>. }
10056   }
10057   {
10058     The~names~already~defined~in~this~document~are:~
10059     \seq_use:Nnnn \g_@@_names_seq { ~and~ } { ,~ } { ~and~ }.
10060   }

10061   \@@_msg_new:nn { Option~auto~for~columns~width }
10062   {
10063     Erroneous~use.\\
10064     You~can't~give~the~value~'auto'~to~the~key~'columns~width'~here.~
10065     That~key~will~be~ignored.
10066   }

10067   \@@_msg_new:nn { NiceTabularX~without~X }
10068   {
10069     NiceTabularX~without~X.\\
10070     You~should~not~use~{NiceTabularX}~without~X~columns.\\
10071     However,~you~can~go~on.
10072   }

10073   \@@_msg_new:nn { Preamble~forgotten }
10074   {
10075     Preamble~forgotten.\\
10076     You~have~probably~forgotten~the~preamble~of~your~
10077     \@@_full_name_env:. \\
10078     This~error~is~fatal.
10079   }

```

Contents

1	Declaration of the package and packages loaded	1
2	Security test	2
3	Collecting options	4
4	Technical definitions	4
5	Parameters	9
6	The command <code>\tabularnote</code>	19
7	Command for creation of rectangle nodes	24
8	The options	25
9	Important code used by <code>{NiceArrayWithDelims}</code>	35
10	The <code>\CodeBefore</code>	49
11	The environment <code>{NiceArrayWithDelims}</code>	52
12	We construct the preamble of the array	57
13	The redefinition of <code>\multicolumn</code>	72
14	The environment <code>{NiceMatrix}</code> and its variants	90
15	<code>{NiceTabular}</code> , <code>{NiceTabularX}</code> and <code>{NiceTabular*}</code>	91
16	After the construction of the array	92
17	We draw the dotted lines	99
18	The actual instructions for drawing the dotted lines with Tikz	112
19	User commands available in the new environments	117
20	The command <code>\line</code> accessible in code-after	123
21	The command <code>\RowStyle</code>	125
22	Colors of cells, rows and columns	128
23	The vertical and horizontal rules	140
24	The empty corners	155
25	The environment <code>{NiceMatrixBlock}</code>	157
26	The extra nodes	158
27	The blocks	163
28	How to draw the dotted lines transparently	183
29	Automatic arrays	183
30	The redefinition of the command <code>\dotfill</code>	184

31	The command <code>\diagbox</code>	185
32	The keyword <code>\CodeAfter</code>	186
33	The delimiters in the preamble	187
34	The command <code>\SubMatrix</code>	188
35	Les commandes <code>\UnderBrace</code> et <code>\OverBrace</code>	196
36	The command <code>TikzEveryCell</code>	199
37	The command <code>\ShowCellNames</code>	201
38	We process the options at package loading	204
39	About the package underscore	205
40	Error messages of the package	206