

The code of the package `nicematrix`*

F. Pantigny
fpantigny@wanadoo.fr

August 14, 2023

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 6.22 of `nicematrix`, at the date of 2023/08/14.

```

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 x }
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:Vn \c_sys_jobname_str { _region_ } % for Emacs
31   || \str_if_eq_p:Vn \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.

```
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 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).

```
114 \tl_new:N \l_@@_argspec_tl
115 \cs_generate_variant:Nn \seq_set_split:Nnn { N V n }
116 \cs_generate_variant:Nn \keys_define:nn { n x }
117 \cs_generate_variant:Nn \str_lowercase:n { V }
118 \hook_gput_code:nnn { begindocument } { . }
119   {
120     \IfPackageLoadedTF { tikz }
121     {
```

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.

```

122     \tl_const:Nn \c_@@_pgfortikzpicture_tl { \exp_not:N \tikzpicture }
123     \tl_const:Nn \c_@@_endpgfortikzpicture_tl { \exp_not:N \endtikzpicture }
124   }
125   {
126     \tl_const:Nn \c_@@_pgfortikzpicture_tl { \exp_not:N \pgfpicture }
127     \tl_const:Nn \c_@@_endpgfortikzpicture_tl { \exp_not:N \endpgfpicture }
128   }
129 }

```

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

```

130 \ifclassloaded { revtex4-1 }
131 { \bool_const:Nn \c_@@_revtex_bool \c_true_bool }
132 {
133   \ifclassloaded { revtex4-2 }
134   { \bool_const:Nn \c_@@_revtex_bool \c_true_bool }
135   {

```

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

```

136     \cs_if_exist:NT \rvtx@ifformat@geq
137     { \bool_const:Nn \c_@@_revtex_bool \c_true_bool }
138     { \bool_const:Nn \c_@@_revtex_bool \c_false_bool }
139   }
140 }

```

```

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

```

The following regex will be used to modify the preamble of the `array` when the key `color-inside` is used.

```

142 \regex_const:Nn \c_@@_columncolor_regex { \c { columncolor } }

```

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.

```

143 \cs_new_protected:Npn \@@_provide_pgfsyspdfmark:
144 {
145   \iow_now:Nn \@mainaux
146   {
147     \ExplSyntaxOn
148     \cs_if_free:NT \pgfsyspdfmark
149     { \cs_set_eq:NN \pgfsyspdfmark \@gobblethree }
150     \ExplSyntaxOff
151   }
152   \cs_gset_eq:NN \@@_provide_pgfsyspdfmark: \prg_do_nothing:
153 }

```

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.

```

154 \ProvideDocumentCommand \iddots { }
155 {
156   \mathinner
157   {
158     \tex_mkern:D 1 mu

```

```

159     \box_move_up:nn { 1 pt } { \hbox:n { . } }
160     \tex_mkern:D 2 mu
161     \box_move_up:nn { 4 pt } { \hbox:n { . } }
162     \tex_mkern:D 2 mu
163     \box_move_up:nn { 7 pt }
164     { \vbox:n { \kern 7 pt \hbox:n { . } } }
165     \tex_mkern:D 1 mu
166   }
167 }

```

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.

```

168 \hook_gput_code:nnn { begindocument } { . }
169 {
170   \IfPackageLoadedTF { booktabs }
171   { \iow_now:Nn \@mainaux \nicematrix@redefine@check@rerun }
172   { }
173 }
174 \cs_set_protected:Npn \nicematrix@redefine@check@rerun
175 {
176   \cs_set_eq:NN \@_old_pgfulil@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`).

```

177   \cs_set_protected:Npn \pgfutil@check@rerun ##1 ##2
178   {
179     \str_if_eq:eeF { nm- } { \tl_range:nnn { ##1 } 1 3 }
180     { \@_old_pgfulil@check@rerun { ##1 } { ##2 } }
181   }
182 }

```

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

```

183 \hook_gput_code:nnn { begindocument } { . }
184 {
185   \IfPackageLoadedTF { colortbl }
186   { }
187   {

```

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.

```

188     \cs_set_protected:Npn \CT@arc@ { }
189     \cs_set:Npn \arrayrulecolor #1 # { \CT@arc { #1 } }
190     \cs_set:Npn \CT@arc #1 #2
191     {
192       \dim_compare:nNnT \baselineskip = \c_zero_dim \noalign
193       { \cs_gset:Npn \CT@arc@ { \color #1 { #2 } } }
194     }

```

Idem for `\CT@drs@`.

```

195     \cs_set:Npn \doublerulesepcolor #1 # { \CT@drs { #1 } }
196     \cs_set:Npn \CT@drs #1 #2
197     {
198       \dim_compare:nNnT \baselineskip = \c_zero_dim \noalign
199       { \cs_gset:Npn \CT@drsc@ { \color #1 { #2 } } }
200     }
201     \cs_set:Npn \hline
202     {
203       \noalign { \ifnum 0 = ` } \fi
204       \cs_set_eq:NN \hskip \vskip
205       \cs_set_eq:NN \vrule \hrule

```

```

206         \cs_set_eq:NN \@width \@height
207         { \CT@arc@ \vline }
208         \futurelet \reserved@a
209         \@xhline
210     }
211 }
212 }

```

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.

```

213 \cs_set:Npn \@@_standard_cline #1 { \@@_standard_cline:w #1 \q_stop }
214 \cs_set:Npn \@@_standard_cline:w #1-#2 \q_stop
215 {
216     \int_compare:nNnT \l_@@_first_col_int = 0 { \omit & }
217     \int_compare:nNnT { #1 } > 1 { \multispan { \int_eval:n { #1 - 1 } } & }
218     \multispan { \int_eval:n { #2 - #1 + 1 } }
219     {
220         \CT@arc@
221         \leaders \hrule \@height \arrayrulewidth \hfill

```

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

```

222     \skip_horizontal:N \c_zero_dim
223 }

```

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`.

```

224     \everycr { }
225     \cr
226     \noalign { \skip_vertical:N -\arrayrulewidth }
227 }

```

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.

```

228 \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`.

```

229 { \@@_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*.

```

230 \cs_set:Npn \@@_cline_i:nn #1 #2 { \@@_cline_i:w #1|#2- \q_stop }
231 \cs_set:Npn \@@_cline_i:w #1|#2-#3 \q_stop
232 {
233     \tl_if_empty:nTF { #3 }
234     { \@@_cline_iii:w #1|#2-#2 \q_stop }
235     { \@@_cline_ii:w #1|#2-#3 \q_stop }
236 }
237 \cs_set:Npn \@@_cline_ii:w #1|#2-#3-\q_stop
238 { \@@_cline_iii:w #1|#2-#3 \q_stop }
239 \cs_set:Npn \@@_cline_iii:w #1|#2-#3 \q_stop
240 {

```

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).

```

241     \int_compare:nNnT { #1 } < { #2 }
242     { \multispan { \int_eval:n { #2 - #1 } } & }

```

¹See question 99041 on TeX StackExchange.

```

243 \multispan { \int_eval:n { #3 - #2 + 1 } }
244 {
245   \CT@arc@
246   \leaders \hrule \@height \arrayrulewidth \hfill
247   \skip_horizontal:N \c_zero_dim
248 }

```

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

```

249 \peek_meaning_remove_ignore_spaces:NTF \cline
250 { & \@@_cline_i:en { \int_eval:n { #3 + 1 } } }
251 { \everycr { } \cr }
252 }
253 \cs_generate_variant:Nn \@@_cline_i:n { e n }

```

The following command is a small shortcut.

```

254 \cs_new:Npn \@@_math_toggle_token:
255 { \bool_if:NF \l_@@_tabular_bool \c_math_toggle_token }

```

```

256 \cs_new_protected:Npn \@@_set_CT@arc@:n #1
257 {
258   \tl_if_blank:nF { #1 }
259   {
260     \tl_if_head_eq_meaning:nNTF { #1 } [
261       { \cs_set:Npn \CT@arc@ { \color #1 } }
262       { \cs_set:Npn \CT@arc@ { \color { #1 } } }
263     ]
264   }
265 \cs_generate_variant:Nn \@@_set_CT@arc@:n { V }

```

```

266 \cs_new_protected:Npn \@@_set_CT@drsc@:n #1
267 {
268   \tl_if_head_eq_meaning:nNTF { #1 } [
269     { \cs_set:Npn \CT@drsc@ { \color #1 } }
270     { \cs_set:Npn \CT@drsc@ { \color { #1 } } }
271   ]
272 \cs_generate_variant:Nn \@@_set_CT@drsc@:n { V }

```

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

```

273 \cs_new:Npn \@@_exp_color_arg:Nn #1 #2
274 {
275   \tl_if_head_eq_meaning:nNTF { #2 } [
276     { #1 #2 }
277     { #1 { #2 } }
278   ]
279 \cs_generate_variant:Nn \@@_exp_color_arg:Nn { N V }

```

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

```

280 \cs_new_protected:Npn \@@_color:n #1
281 { \tl_if_blank:nF { #1 } { \@@_exp_color_arg:Nn \color { #1 } } }
282 \cs_generate_variant:Nn \@@_color:n { V }

```

```

283 \cs_set_eq:NN \@@_old_pgfpntanchor \pgfpntanchor

```

The column S of siunitx

The command `\@@_renew_NC@rewrite@S:` will be used in each environment of `nicematrix` in order to “rewrite” the S column in each environment.

```

284 \hook_gput_code:nnn { begindocument } { . }
285 {

```



```

286 \IfPackageLoadedTF { siunitx }
287 {
288   \cs_new_protected:Npn \@@_renew_NC@rewrite@S:
289   {
290     \renewcommand*{\NC@rewrite@S}[1] []
291     {
\@temptokena is a toks (not supported by the L3 programming layer).
292       \tl_if_empty:nTF { ##1 }
293       {
294         \@temptokena \exp_after:wN
295         { \tex_the:D \@temptokena \@@_S: }
296       }
297       {
298         \@temptokena \exp_after:wN
299         { \tex_the:D \@temptokena \@@_S: [ ##1 ] }
300       }
301       \NC@find
302     }
303   }
304 }
305 { \cs_set_eq:NN \@@_renew_NC@rewrite@S: \prg_do_nothing: }
306 }

307 \cs_new_protected:Npn \@@_rescan_for_spanish:N #1
308 {
309   \tl_set_rescan:Nno
310   #1
311   {
312     \char_set_catcode_other:N >
313     \char_set_catcode_other:N <
314   }
315   #1
316 }

```

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.

```
317 \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).

```
318 \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.

```
319 \NewExpandableDocumentCommand \NiceMatrixLastEnv { }
320 { \int_use:N \g_@@_env_int }
```

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

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

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

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

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

```
324 \bool_new:N \g_@@_delims_bool
325 \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 an environment (provided by the final user): `{NiceTabular}`, `{NiceArray}`, `{pNiceArray}`, etc.

```
326 \bool_new:N \l_@@_preamble_bool
327 \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.

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

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

```
329 \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.

```
330 \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).

```
331 \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.).

```
332 \dim_new:N \l_@@_col_width_dim
333 \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.

```
334 \int_new:N \g_@@_row_total_int
335 \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).

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

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

```
337 \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.

```
338 \str_new:N \l_@@_hpos_cell_str
339 \str_set:Nn \l_@@_hpos_cell_str { c }
```

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`.

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

Idem for the mono-row blocks.

```
341 \dim_new:N \g_@@_blocks_ht_dim
```

```
342 \dim_new:N \g_@@_blocks_dp_dim
```

The following dimension will be used by the command `\Block` for the blocks with a key of vertical position equal to T or B.

```
343 \dim_new:N \l_@@_block_ysep_dim
```

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

```
344 \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`.

```
345 \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.

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

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

```
347 \bool_new:N \l_@@_notes_detect_duplicates_bool
```

```
348 \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.

```
349 \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).

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

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

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

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

```
352 \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.

```
353 \bool_new:N \l_@@_X_column_bool
```

```
354 \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 contain 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 }`).

```
355 \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.

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

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

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

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

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

```
360 \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`. However, it does *not* contain the value provided by the final user. Indeed, a transformation is done in order to have a preamble (for the package `array`) which is `nicematrix`-aware. That transformation is done with the command `\@@_set_preamble:Nn`.

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

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

```
363 { \@@_set_preamble: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 `.`.

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

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

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

```
367 \cs_new_protected:Npn \@@_test_if_math_mode:
```

```
368 {
```

```
369   \if_mode_math: \else:
```

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

```
371   \fi:
```

```
372 }
```

The letter used for the vlines which will be drawn only in the sub-matrices. `vlism` stands for *vertical lines in sub-matrices*.

```
373 \tl_new:N \l_@@_letter_vlism_tl
```

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.

```
374 \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”.

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

```
376 \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`).

```
377 \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*.

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

```
379 \tl_gset:Nn \g_@@_com_or_env_str { environment }
```

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).

```

380 \cs_new:Npn \@@_full_name_env:
381   {
382     \str_if_eq:VnTF \g_@@_com_or_env_str { command }
383       { command \space \c_backslash_str \g_@@_name_env_str }
384     { environment \space \{ \g_@@_name_env_str \} }
385   }

```

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

```

386 \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.

```

387 \tl_new:N \l_@@_pgf_node_code_tl

```

The following token list has a function similar to `\g_nicematrix_code_after_tl` but it is used internally by `nicematrix`. In fact, we have to distinguish between `\g_nicematrix_code_after_tl` and `\g_@@_pre_code_after_tl` because we must take care of the order in which instructions stored in that parameters are executed.

```

388

```

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

```

389 \tl_new:N \g_@@_pre_code_before_tl
390 \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.

```

391 \tl_new:N \g_@@_pre_code_after_tl
392 \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.

```

393 \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.

```

394 \int_new:N \l_@@_old_iRow_int
395 \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).

```

396 \seq_new:N \l_@@_custom_line_commands_seq

```

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

```

397 \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.

```
398 \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.

```
399 \bool_new:N \l_@@_X_columns_aux_bool
400 \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.

```
401 \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).

```
402 \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).

```
403 \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`).

```
404 \tl_new:N \l_@@_code_before_tl
405 \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).

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

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

```
407 \dim_new:N \l_@@_x_initial_dim
408 \dim_new:N \l_@@_y_initial_dim
409 \dim_new:N \l_@@_x_final_dim
410 \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.

```
411 \dim_zero_new:N \l_@@_tmpc_dim
412 \dim_zero_new:N \l_@@_tmpd_dim
```

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

```
413 \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”.

```
414 \dim_new:N \g_@@_width_last_col_dim
415 \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.

```
416 \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!).

```
417 \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}`.

```
418 \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.

```
419 \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.

```
420 \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`).

```
421 \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.

```
422 \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).

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

```
424 \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`).

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

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

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

```
428 \int_new:N \l_@@_col_max_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.

```
429 \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).

```
430 \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`.

```
431 \tl_new:N \l_@@_fill_tl
432 \tl_new:N \l_@@_opacity_tl
433 \tl_new:N \l_@@_draw_tl
434 \seq_new:N \l_@@_tikz_seq
435 \clist_new:N \l_@@_borders_clist
436 \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.

```
437 \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`.

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

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

```
439 \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`).

```
440 \str_new:N \l_@@_hpos_block_str
441 \str_set:Nn \l_@@_hpos_block_str { c }
442 \bool_new:N \l_@@_hpos_of_block_cap_bool
```

For the vertical position, the possible values are `c`, `t` and `b`. Of course, it would be interesting to program a key `T` and a key `B`.

```
443 \str_new:N \l_@@_vpos_of_block_str
444 \str_set:Nn \l_@@_vpos_of_block_str { c }
```

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

```
445 \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).

```
446 \bool_new:N \l_@@_vlines_block_bool
447 \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.

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



```

449 \dim_new:N \l_@@_submatrix_extra_height_dim
450 \dim_new:N \l_@@_submatrix_left_xshift_dim
451 \dim_new:N \l_@@_submatrix_right_xshift_dim
452 \clist_new:N \l_@@_hlines_clist
453 \clist_new:N \l_@@_vlines_clist
454 \clist_new:N \l_@@_submatrix_hlines_clist
455 \clist_new:N \l_@@_submatrix_vlines_clist

```

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

```

456 \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.

```

457 \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`).

```

458 \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.

```

459 \int_new:N \l_@@_first_row_int
460 \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.

```

461 \int_new:N \l_@@_first_col_int
462 \int_set:Nn \l_@@_first_col_int 1

```

- **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).

```

463 \int_new:N \l_@@_last_row_int
464 \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

```

465 \bool_new:N \l_@@_last_row_without_value_bool

```

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

Idem for `\l_@@_last_col_without_value_bool`

```
466 \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.

```
467 \int_new:N \l_@@_last_col_int
468 \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.

```
469 \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`).

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

Some utilities

```
471 \cs_set_protected:Npn \@@_cut_on_hyphen:w #1-#2\q_stop
472 {
473 \tl_set:Nn \l_tmpa_tl { #1 }
474 \tl_set:Nn \l_tmpb_tl { #2 }
475 }
```

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.

```
476 \cs_new_protected:Npn \@@_expand_clist:N #1
477 {
478 \clist_if_in:NnF #1 { all }
479 {
480 \clist_clear:N \l_tmpa_clist
481 \clist_map_inline:Nn #1
482 {
483 \tl_if_in:nnTF { ##1 } { - }
484 { \@@_cut_on_hyphen:w ##1 \q_stop }
485 {
486 \tl_set:Nn \l_tmpa_tl { ##1 }
487 \tl_set:Nn \l_tmpb_tl { ##1 }
488 }
489 \int_step_inline:nnn { \l_tmpa_tl } { \l_tmpb_tl }
490 { \clist_put_right:Nn \l_tmpa_clist { ####1 } }
491 }
492 \tl_set_eq:NN #1 \l_tmpa_clist
493 }
494 }
```

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.

```

495 \hook_gput_code:nnn { begindocument } { . }
496 {
497   \dim_const:Nn \c_@@_shift_Ldots_last_row_dim { 0.5 em }
498   \dim_const:Nn \c_@@_shift_exterior_Vdots_dim { 0.6 em }
499   \dim_const:Nn \c_@@_innersep_middle_dim { 0.17 em }
500 }
```

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 composant of `\g_@@_notes_seq` will be a kind of couple of the form : `{label}{text of the tabularnote}`. The first composante 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 `\c_novalue_tl`).
- During the composition of the caption (value of `\l_@@_caption_tl`), the tabular notes will be numbered from 1 to `\g_@@_notes_caption_int` and the notes themselves will be stored in `\g_@@_notes_in_caption_seq`. The structure of the composantes 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.

```

501 \newcounter { tabularnote }
```

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

```

502 \seq_new:N \g_@@_notes_seq
503 \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.

```

504 \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.

```

505 \seq_new:N \l_@@_notes_labels_seq
506 \newcounter{nicematrix_draft}
507 \cs_new_protected:Npn \@@_notes_format:n #1
508 {
509   \setcounter { nicematrix_draft } { #1 }
510   \@@_notes_style:n { nicematrix_draft }
511 }

```

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

```

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

```

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

```

513 \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`.

```

514 \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`.

```

515 \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`).

```

516 \hook_gput_code:nnn { begindocument } { . }
517 {
518   \IfPackageLoadedTF { enumitem }
519   {

```

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.

```

520   \newlist { tabularnotes } { enumerate } { 1 }
521   \setlist [ tabularnotes ]
522   {
523     topsep = 0pt ,
524     noitemsep ,
525     leftmargin = * ,
526     align = left ,
527     labelsep = 0pt ,
528     label =
529     \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotesi } } ,
530   }
531   \newlist { tabularnotes* } { enumerate* } { 1 }
532   \setlist [ tabularnotes* ]
533   {
534     afterlabel = \nobreak ,

```

```

535     itemjoin = \quad ,
536     label =
537     \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotes*i } }
538 }

```

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`.

```

539     \NewDocumentCommand \tabularnote { o m }
540     {
541         \bool_if:nT { \cs_if_exist_p:N \@capttype || \l_@@_in_env_bool }
542         {
543             \bool_if:nTF { ! \l_@@_tabular_bool && \l_@@_in_env_bool }
544             { \@@_error:n { tabularnote~forbidden } }
545             {
546                 \bool_if:NTF \l_@@_in_caption_bool
547                 { \@@_tabularnote_caption:nn { #1 } { #2 } }
548                 { \@@_tabularnote:nn { #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 }

```

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 `\c_novalue_tl`) and #2 is the mandatory argument of `\tabularnote`.

```

561 \cs_new_protected:Npn \@@_tabularnote:nn #1 #2
562 {

```

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).

```

563     \int_zero:N \l_tmpa_int
564     \bool_if:NT \l_@@_notes_detect_duplicates_bool
565     {

```

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

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

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

```

566     \seq_map_indexed_inline:Nn \g_@@_notes_seq
567     {
568         \tl_if_eq:nnT { { #1 } { #2 } } { ##2 }
569         {
570             \int_set:Nn \l_tmpa_int { ##1 }
571             \seq_map_break:
572         }
573     }
574     \int_compare:nNnF \l_tmpa_int = \c_zero_int
575     { \int_add:Nn \l_tmpa_int \g_@@_notes_caption_int }

```

```

576     }
577     \int_compare:nNnT \l_tmpa_int = \c_zero_int
578     {
579         \seq_gput_right:Nn \g_@@_notes_seq { { #1 } { #2 } }
580         \tl_if_novalue:nT { #1 } { \int_gincr:N \c@tabularnote }
581     }
582     \seq_put_right:Nx \l_@@_notes_labels_seq
583     {
584         \tl_if_novalue:nTF { #1 }
585         {
586             \@@_notes_format:n
587             {
588                 \int_eval:n
589                 {
590                     \int_compare:nNnTF \l_tmpa_int = \c_zero_int
591                     \c@tabularnote
592                     \l_tmpa_int
593                 }
594             }
595         }
596         { #1 }
597     }
598     \peek_meaning:NF \tabularnote
599     {

```

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

```

600         \hbox_set:Nn \l_tmpa_box
601         {

```

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

```

602             \@@_notes_label_in_tabular:n
603             {
604                 \seq_use:Nnnn
605                 \l_@@_notes_labels_seq { , } { , } { , }
606             }
607         }

```

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

```

608         \int_gsub:Nn \c@tabularnote { 1 }
609         \int_set_eq:NN \l_tmpa_int \c@tabularnote
610         \refstepcounter { tabularnote }
611         \int_compare:nNnT \l_tmpa_int = \c@tabularnote
612         { \int_gincr:N \c@tabularnote }
613         \seq_clear:N \l_@@_notes_labels_seq
614         \bool_lazy_or:nnTF
615         { \str_if_eq_p:Vn \l_@@_hpos_cell_str { c } }
616         { \str_if_eq_p:Vn \l_@@_hpos_cell_str { r } }
617         {
618             \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).

```

619             \skip_horizontal:n { \box_wd:N \l_tmpa_box }
620         }
621         { \box_use:N \l_tmpa_box }
622     }
623 }

```

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`.

```

624 \cs_new_protected:Npn \@@_tabularnote_caption:nn #1 #2
625   {
626     \bool_if:NTF \g_@@_caption_finished_bool
627       {
628         \int_compare:nNnT
629           \c@tabularnote = \g_@@_notes_caption_int
630           { \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`!

```

631     \seq_if_in:NnF \g_@@_notes_in_caption_seq { { #1 } { #2 } }
632     { \@@_error:n { Identical~notes~in~caption } }
633   }
634   {

```

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

```

635     \seq_if_in:NnTF \g_@@_notes_in_caption_seq { { #1 } { #2 } }
636     {

```

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.

```

637         \bool_gset_true:N \g_@@_caption_finished_bool
638         \int_gset_eq:NN \g_@@_notes_caption_int \c@tabularnote
639         \int_gzero:N \c@tabularnote
640       }
641       { \seq_gput_right:Nn \g_@@_notes_in_caption_seq { { #1 } { #2 } } }
642     }

```

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!

```

643     \tl_if_novalue:nT { #1 } { \int_gincr:N \c@tabularnote }
644     \seq_put_right:Nx \l_@@_notes_labels_seq
645     {
646       \tl_if_novalue:nTF { #1 }
647       { \@@_notes_format:n { \int_use:N \c@tabularnote } }
648       { #1 }
649     }
650     \peek_meaning:NF \tabularnote
651     {
652       \@@_notes_label_in_tabular:n
653       { \seq_use:Nnnn \l_@@_notes_labels_seq { , } { , } { , } }
654       \seq_clear:N \l_@@_notes_labels_seq
655     }
656   }

657 \cs_new_protected:Npn \@@_count_novalue_first:nn #1 #2
658   { \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.

```

659 \cs_new_protected:Npn \@@_pgf_rect_node:nnnnn #1 #2 #3 #4 #5
660 {
661   \begin { pgfscope }
662   \pgfset
663   {
664     inner~sep = \c_zero_dim ,
665     minimum~size = \c_zero_dim
666   }
667   \pgftransformshift { \pgfpoint { 0.5 * ( #2 + #4 ) } { 0.5 * ( #3 + #5 ) } }
668   \pgfnode
669   { rectangle }
670   { center }
671   {
672     \vbox_to_ht:nn
673     { \dim_abs:n { #5 - #3 } }
674     {
675       \vfill
676       \hbox_to_wd:nn { \dim_abs:n { #4 - #2 } } { }
677     }
678   }
679   { #1 }
680   { }
681   \end { pgfscope }
682 }

```

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.

```

683 \cs_new_protected:Npn \@@_pgf_rect_node:nnn #1 #2 #3
684 {
685   \begin { pgfscope }
686   \pgfset
687   {
688     inner~sep = \c_zero_dim ,
689     minimum~size = \c_zero_dim
690   }
691   \pgftransformshift { \pgfpointscale { 0.5 } { \pgfpointadd { #2 } { #3 } } }
692   \pgfpointdiff { #3 } { #2 }
693   \pgfgetlastxy \l_tmpa_dim \l_tmpb_dim
694   \pgfnode
695   { rectangle }
696   { center }
697   {
698     \vbox_to_ht:nn
699     { \dim_abs:n \l_tmpb_dim }
700     { \vfill \hbox_to_wd:nn { \dim_abs:n \l_tmpa_dim } { } }
701   }
702   { #1 }
703   { }
704   \end { pgfscope }
705 }

```

8 The options

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

```

706 \tl_new:N \l_@@_caption_tl

```



```

707 \tl_new:N \l_@@_short_caption_tl
708 \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).

```

709 \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.

```

710 \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`.

```

711 \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`).

```

712 \dim_new:N \l_@@_cell_space_top_limit_dim
713 \dim_new:N \l_@@_cell_space_bottom_limit_dim

```

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

```

714 \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.

```

715 \dim_new:N \l_@@_xdots_inter_dim
716 \hook_gput_code:nnn { begindocument } { . }
717   { \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).

```

718 \dim_new:N \l_@@_xdots_shorten_start_dim
719 \dim_new:N \l_@@_xdots_shorten_end_dim
720 \hook_gput_code:nnn { begindocument } { . }
721   {
722     \dim_set:Nn \l_@@_xdots_shorten_start_dim { 0.3 em }
723     \dim_set:Nn \l_@@_xdots_shorten_end_dim { 0.3 em }
724   }

```

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.

```

725 \dim_new:N \l_@@_xdots_radius_dim
726 \hook_gput_code:nnn { begindocument } { . }
727   { \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.

```
728 \tl_new:N \l_@@_xdots_line_style_tl
729 \tl_const:Nn \c_@@_standard_tl { standard }
730 \tl_set_eq:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl
```

The boolean `\l_@@_light_syntax_bool` corresponds to the option `light-syntax`.

```
731 \bool_new:N \l_@@_light_syntax_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).

```
732 \tl_new:N \l_@@_baseline_tl
733 \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`).

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

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

```
735 \bool_new:N \l_@@_parallelize_diags_bool
736 \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`.

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

738 \dim_new:N \l_@@_notes_above_space_dim
739 \hook_gput_code:nnn { begindocument } { . }
740 { \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.).

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

The following flag corresponds to the key `respect-arraystretch` (that key has an effect on the blocks).

```
742 \bool_new:N \l_@@_respect_arraystretch_bool
```

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).

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

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

```
744 \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.

```
745 \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”.

```
746 \bool_new:N \l_@@_medium_nodes_bool
```

```
747 \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).

```
748 \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`).

```
749 \dim_new:N \l_@@_left_margin_dim
```

```
750 \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`.

```
751 \dim_new:N \l_@@_extra_left_margin_dim
```

```
752 \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.

```
753 \tl_new:N \l_@@_end_of_row_tl
```

```
754 \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 “:”.

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

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

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

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

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

```
758 % \bigskip
```

```
759 % \begin{macrocode}
```

```
760 \keys_define:nn { NiceMatrix / xdots }
```

```
761 {
```

```
762   shorten-start .code:n =
```

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

```
764     { \dim_set:Nn \l_@@_xdots_shorten_start_dim { #1 } } ,
```

```
765   shorten-end .code:n =
```

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

```
767     { \dim_set:Nn \l_@@_xdots_shorten_end_dim { #1 } } ,
```

```
768   shorten-start .value_required:n = true ,
```

```
769   shorten-end .value_required:n = true ,
```

```

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

```

798 down .code:n = \tl_put_right:Nn \l_@@_xdots_down_tl { #1 } , % modified 2023-08-09
799 up .code:n = \tl_put_right:Nn \l_@@_xdots_up_tl { #1 } ,
800 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).

```

801 draw-first .code:n = \prg_do_nothing: ,
802 unknown .code:n = \@@_error:n { Unknown-key-for~xdots }
803 }

```

```

804 \keys_define:nn { NiceMatrix / rules }
805 {
806   color .tl_set:N = \l_@@_rules_color_tl ,
807   color .value_required:n = true ,
808   width .dim_set:N = \arrayrulewidth ,
809   width .value_required:n = true ,
810   unknown .code:n = \@@_error:n { Unknown~key-for~rules }
811 }

```

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

```

812 \keys_define:nn { NiceMatrix / Global }
813 {
814   rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
815   rounded-corners .default:n = 4 pt ,
816   custom-line .code:n = \@@_custom_line:n { #1 } ,
817   rules .code:n = \keys_set:nn { NiceMatrix / rules } { #1 } ,
818   rules .value_required:n = true ,
819   standard-cline .bool_set:N = \l_@@_standard_cline_bool ,

```

```

820 standard-cline .default:n = true ,
821 cell-space-top-limit .dim_set:N = \l_@@_cell_space_top_limit_dim ,
822 cell-space-top-limit .value_required:n = true ,
823 cell-space-bottom-limit .dim_set:N = \l_@@_cell_space_bottom_limit_dim ,
824 cell-space-bottom-limit .value_required:n = true ,
825 cell-space-limits .meta:n =
826 {
827   cell-space-top-limit = #1 ,
828   cell-space-bottom-limit = #1 ,
829 } ,
830 cell-space-limits .value_required:n = true ,
831 xdots .code:n = \keys_set:mn { NiceMatrix / xdots } { #1 } ,
832 light-syntax .bool_set:N = \l_@@_light_syntax_bool ,
833 light-syntax .default:n = true ,
834 end-of-row .tl_set:N = \l_@@_end_of_row_tl ,
835 end-of-row .value_required:n = true ,
836 first-col .code:n = \int_zero:N \l_@@_first_col_int ,
837 first-row .code:n = \int_zero:N \l_@@_first_row_int ,
838 last-row .int_set:N = \l_@@_last_row_int ,
839 last-row .default:n = -1 ,
840 code-for-first-col .tl_set:N = \l_@@_code_for_first_col_tl ,
841 code-for-first-col .value_required:n = true ,
842 code-for-last-col .tl_set:N = \l_@@_code_for_last_col_tl ,
843 code-for-last-col .value_required:n = true ,
844 code-for-first-row .tl_set:N = \l_@@_code_for_first_row_tl ,
845 code-for-first-row .value_required:n = true ,
846 code-for-last-row .tl_set:N = \l_@@_code_for_last_row_tl ,
847 code-for-last-row .value_required:n = true ,
848 hlines .clist_set:N = \l_@@_hlines_clist ,
849 vlines .clist_set:N = \l_@@_vlines_clist ,
850 hlines .default:n = all ,
851 vlines .default:n = all ,
852 vlines-in-sub-matrix .code:n =
853 {
854   \tl_if_single_token:nTF { #1 }
855     { \tl_set:Nn \l_@@_letter_vlism_tl { #1 } }
856     { \@@_error:n { One-letter~allowed } }
857 } ,
858 vlines-in-sub-matrix .value_required:n = true ,
859 hvlines .code:n =
860 {
861   \bool_set_true:N \l_@@_hvlines_bool
862   \clist_set:Nn \l_@@_vlines_clist { all }
863   \clist_set:Nn \l_@@_hlines_clist { all }
864 } ,
865 hvlines-except-borders .code:n =
866 {
867   \clist_set:Nn \l_@@_vlines_clist { all }
868   \clist_set:Nn \l_@@_hlines_clist { all }
869   \bool_set_true:N \l_@@_hvlines_bool
870   \bool_set_true:N \l_@@_except_borders_bool
871 } ,
872 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.

```

873 renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
874 renew-dots .value_forbidden:n = true ,
875 nullify-dots .bool_set:N = \l_@@_nullify_dots_bool ,
876 create-medium-nodes .bool_set:N = \l_@@_medium_nodes_bool ,
877 create-large-nodes .bool_set:N = \l_@@_large_nodes_bool ,
878 create-extra-nodes .meta:n =
879 { create-medium-nodes , create-large-nodes } ,

```

```

880 left-margin .dim_set:N = \l_@@_left_margin_dim ,
881 left-margin .default:n = \arraycolsep ,
882 right-margin .dim_set:N = \l_@@_right_margin_dim ,
883 right-margin .default:n = \arraycolsep ,
884 margin .meta:n = { left-margin = #1 , right-margin = #1 } ,
885 margin .default:n = \arraycolsep ,
886 extra-left-margin .dim_set:N = \l_@@_extra_left_margin_dim ,
887 extra-right-margin .dim_set:N = \l_@@_extra_right_margin_dim ,
888 extra-margin .meta:n =
889   { extra-left-margin = #1 , extra-right-margin = #1 } ,
890 extra-margin .value_required:n = true ,
891 respect-arraystretch .bool_set:N = \l_@@_respect_arraystretch_bool ,
892 respect-arraystretch .default:n = true ,
893 pgf-node-code .tl_set:N = \l_@@_pgf_node_code_tl ,
894 pgf-node-code .value_required:n = true
895 }

```

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

```

896 \keys_define:nn { NiceMatrix / Env }
897   {
898     corners .clist_set:N = \l_@@_corners_clist ,
899     corners .default:n = { NW , SW , NE , SE } ,
900     code-before .code:n =
901       {
902         \tl_if_empty:nF { #1 }
903           {
904             \tl_gput_left:Nn \g_@@_pre_code_before_tl { #1 }
905             \bool_set_true:N \l_@@_code_before_bool
906           }
907       } ,
908     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}`.

```

909 c .code:n = \tl_set:Nn \l_@@_baseline_tl c ,
910 t .code:n = \tl_set:Nn \l_@@_baseline_tl t ,
911 b .code:n = \tl_set:Nn \l_@@_baseline_tl b ,
912 baseline .tl_set:N = \l_@@_baseline_tl ,
913 baseline .value_required:n = true ,
914 columns-width .code:n =
915   \tl_if_eq:nnTF { #1 } { auto }
916     { \bool_set_true:N \l_@@_auto_columns_width_bool }
917     { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,
918 columns-width .value_required:n = true ,
919 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.

```

920 \legacy_if:nF { measuring@ }
921   {
922     \str_set:Nx \l_tmpa_str { #1 }
923     \seq_if_in:NVTF \g_@@_names_seq \l_tmpa_str
924       { \@@_error:nn { Duplicate-name } { #1 } }
925       { \seq_gput_left:NV \g_@@_names_seq \l_tmpa_str }
926     \str_set_eq:NN \l_@@_name_str \l_tmpa_str
927   } ,
928 name .value_required:n = true ,
929 code-after .tl_gset:N = \g_nicematrix_code_after_tl ,
930 code-after .value_required:n = true ,
931 color-inside .code:n =
932   \bool_set_true:N \l_@@_color_inside_bool

```

```

933     \bool_set_true:N \l_@@_code_before_bool ,
934     color-inside .value_forbidden:n = true ,
935     colortbl-like .meta:n = color-inside
936   }

937 \keys_define:nn { NiceMatrix / notes }
938   {
939     para .bool_set:N = \l_@@_notes_para_bool ,
940     para .default:n = true ,
941     code-before .tl_set:N = \l_@@_notes_code_before_tl ,
942     code-before .value_required:n = true ,
943     code-after .tl_set:N = \l_@@_notes_code_after_tl ,
944     code-after .value_required:n = true ,
945     bottomrule .bool_set:N = \l_@@_notes_bottomrule_bool ,
946     bottomrule .default:n = true ,
947     style .code:n = \cs_set:Nn \@@_notes_style:n { #1 } ,
948     style .value_required:n = true ,
949     label-in-tabular .code:n =
950       \cs_set:Nn \@@_notes_label_in_tabular:n { #1 } ,
951     label-in-tabular .value_required:n = true ,
952     label-in-list .code:n =
953       \cs_set:Nn \@@_notes_label_in_list:n { #1 } ,
954     label-in-list .value_required:n = true ,
955     enumitem-keys .code:n =
956       {
957         \hook_gput_code:nnn { begindocument } { . }
958         {
959           \IfPackageLoadedTF { enumitem }
960             { \setlist* [ tabularnotes ] { #1 } }
961             { }
962         }
963       } ,
964     enumitem-keys .value_required:n = true ,
965     enumitem-keys-para .code:n =
966       {
967         \hook_gput_code:nnn { begindocument } { . }
968         {
969           \IfPackageLoadedTF { enumitem }
970             { \setlist* [ tabularnotes* ] { #1 } }
971             { }
972         }
973       } ,
974     enumitem-keys-para .value_required:n = true ,
975     detect-duplicates .bool_set:N = \l_@@_notes_detect_duplicates_bool ,
976     detect-duplicates .default:n = true ,
977     unknown .code:n = \@@_error:n { Unknown-key-for-notes }
978   }

979 \keys_define:nn { NiceMatrix / delimiters }
980   {
981     max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
982     max-width .default:n = true ,
983     color .tl_set:N = \l_@@_delimiters_color_tl ,
984     color .value_required:n = true ,
985   }

```

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

```

986 \keys_define:nn { NiceMatrix }
987   {
988     NiceMatrixOptions .inherit:n =
989       { NiceMatrix / Global } ,
990     NiceMatrixOptions / xdots .inherit:n = NiceMatrix / xdots ,
991     NiceMatrixOptions / rules .inherit:n = NiceMatrix / rules ,

```

```

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

```

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

```

1027 \keys_define:nn { NiceMatrix / NiceMatrixOptions }
1028   {
1029     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1030     delimiters / color .value_required:n = true ,
1031     delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1032     delimiters / max-width .default:n = true ,
1033     delimiters .code:n = \keys_set:nn { NiceMatrix / delimiters } { #1 } ,
1034     delimiters .value_required:n = true ,
1035     width .code:n = \dim_set:Nn \l_@@_width_dim { #1 } ,
1036     width .value_required:n = true ,
1037     last-col .code:n =
1038       \tl_if_empty:nF { #1 }
1039         { \@@_error:n { last-col-non-empty-for-NiceMatrixOptions } }
1040         \int_zero:N \l_@@_last_col_int ,
1041     small .bool_set:N = \l_@@_small_bool ,
1042     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.

```

1043     renew-matrix .code:n = \@@_renew_matrix: ,
1044     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}`.

```

1045     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.

```

1046   columns-width .code:n =
1047     \tl_if_eq:nnTF { #1 } { auto }
1048     { \@@_error:n { Option~auto~for~columns~width } }
1049     { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,

```

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

```

1050   allow-duplicate-names .code:n =
1051     \@@_msg_redirect_name:nn { Duplicate-name } { none } ,
1052   allow-duplicate-names .value_forbidden:n = true ,
1053   notes .code:n = \keys_set:nn { NiceMatrix / notes } { #1 } ,
1054   notes .value_required:n = true ,
1055   sub-matrix .code:n = \keys_set:nn { NiceMatrix / sub-matrix } { #1 } ,
1056   sub-matrix .value_required:n = true ,
1057   matrix / columns-type .code:n =
1058     \@@_set_preamble:Nn \l_@@_columns_type_tl { #1 } ,
1059   matrix / columns-type .value_required:n = true ,
1060   caption-above .bool_set:N = \l_@@_caption_above_bool ,
1061   caption-above .default:n = true ,
1062   unknown .code:n = \@@_error:n { Unknown~key~for~NiceMatrixOptions }
1063 }

```

`\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.

```

1064 \NewDocumentCommand \NiceMatrixOptions { m }
1065 { \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.

```

1066 \keys_define:nn { NiceMatrix / NiceMatrix }
1067 {
1068   last-col .code:n = \tl_if_empty:nTF {#1}
1069     {
1070       \bool_set_true:N \l_@@_last_col_without_value_bool
1071       \int_set:Nn \l_@@_last_col_int { -1 }
1072     }
1073     { \int_set:Nn \l_@@_last_col_int { #1 } } ,
1074   columns-type .code:n = \@@_set_preamble:Nn \l_@@_columns_type_tl { #1 } ,
1075   columns-type .value_required:n = true ,
1076   l .meta:n = { columns-type = l } ,
1077   r .meta:n = { columns-type = r } ,
1078   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1079   delimiters / color .value_required:n = true ,
1080   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1081   delimiters / max-width .default:n = true ,
1082   delimiters .code:n = \keys_set:nn { NiceMatrix / delimiters } { #1 } ,
1083   delimiters .value_required:n = true ,
1084   small .bool_set:N = \l_@@_small_bool ,
1085   small .value_forbidden:n = true ,
1086   unknown .code:n = \@@_error:n { Unknown~key~for~NiceMatrix }
1087 }

```

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

```

1088 \keys_define:nn { NiceMatrix / NiceArray }
1089 {

```

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.

```

1090   small .bool_set:N = \l_@@_small_bool ,
1091   small .value_forbidden:n = true ,
1092   last-col .code:n = \tl_if_empty:nF { #1 }
1093             { \@@_error:n { last-col~non-empty~for~NiceArray } }
1094             \int_zero:N \l_@@_last_col_int ,
1095   r .code:n = \@@_error:n { r~or~l~with~preamble } ,
1096   l .code:n = \@@_error:n { r~or~l~with~preamble } ,
1097   unknown .code:n = \@@_error:n { Unknown-key-for-NiceArray }
1098 }
1099 \keys_define:nn { NiceMatrix / pNiceArray }
1100 {
1101   first-col .code:n = \int_zero:N \l_@@_first_col_int ,
1102   last-col .code:n = \tl_if_empty:nF {#1}
1103             { \@@_error:n { last-col~non-empty~for~NiceArray } }
1104             \int_zero:N \l_@@_last_col_int ,
1105   first-row .code:n = \int_zero:N \l_@@_first_row_int ,
1106   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1107   delimiters / color .value_required:n = true ,
1108   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1109   delimiters / max-width .default:n = true ,
1110   delimiters .code:n = \keys_set:nn { NiceMatrix / delimiters } { #1 } ,
1111   delimiters .value_required:n = true ,
1112   small .bool_set:N = \l_@@_small_bool ,
1113   small .value_forbidden:n = true ,
1114   r .code:n = \@@_error:n { r~or~l~with~preamble } ,
1115   l .code:n = \@@_error:n { r~or~l~with~preamble } ,
1116   unknown .code:n = \@@_error:n { Unknown-key-for-NiceMatrix }
1117 }

```

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

```

1118 \keys_define:nn { NiceMatrix / NiceTabular }
1119 {

```

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.

```

1120   width .code:n = \dim_set:Nn \l_@@_width_dim { #1 }
1121             \bool_set_true:N \l_@@_width_used_bool ,
1122   width .value_required:n = true ,
1123   notes .code:n = \keys_set:nn { NiceMatrix / notes } { #1 } ,
1124   tabularnote .tl_gset:N = \g_@@_tabularnote_tl ,
1125   tabularnote .value_required:n = true ,
1126   caption .tl_set:N = \l_@@_caption_tl ,
1127   caption .value_required:n = true ,
1128   short-caption .tl_set:N = \l_@@_short_caption_tl ,
1129   short-caption .value_required:n = true ,
1130   label .tl_set:N = \l_@@_label_tl ,
1131   label .value_required:n = true ,
1132   last-col .code:n = \tl_if_empty:nF {#1}
1133             { \@@_error:n { last-col~non-empty~for~NiceArray } }
1134             \int_zero:N \l_@@_last_col_int ,
1135   r .code:n = \@@_error:n { r~or~l~with~preamble } ,
1136   l .code:n = \@@_error:n { r~or~l~with~preamble } ,
1137   unknown .code:n = \@@_error:n { Unknown-key-for-NiceTabular }
1138 }

```

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}`).

```
1139 \cs_new_protected:Npn \@@_cell_begin:w
1140   {
```

`\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.

```
1141   \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).

```
1142   \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
```

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

```
1143   \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 `arydshn`, create special rows in the `\halign` that we don't want to take into account.

```
1144   \int_compare:nNnT \c@jCol = 1
1145   { \int_compare:nNnT \l_@@_first_col_int = 1 \@@_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` will be in the `\@@_cell_end:` (and the potential `\c_math_toggle_token` also).

```
1146   \hbox_set:Nw \l_@@_cell_box
1147   \bool_if:NF \l_@@_tabular_bool
1148   {
1149     \c_math_toggle_token
1150     \bool_if:NT \l_@@_small_bool \scriptstyle
1151   }
1152   \g_@@_row_style_tl
```

We will call *corners* of the matrix the cases which are at the intersection of the exterior rows and exterior columns (of course, the four corners doesn't always exist simultaneously).

The codes `\l_@@_code_for_first_row_tl` and *al* don't apply in the corners of the matrix.

```
1153   \int_compare:nNnTF \c@iRow = 0
1154   {
1155     \int_compare:nNnT \c@jCol > 0
1156     {
1157       \l_@@_code_for_first_row_tl
1158       \xglobal \colorlet { nicematrix-first-row } { . }
1159     }
1160   }
1161   {
1162     \int_compare:nNnT \c@iRow = \l_@@_last_row_int
1163     {
1164       \l_@@_code_for_last_row_tl
1165       \xglobal \colorlet { nicematrix-last-row } { . }
1166     }
1167   }
1168 }
```

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.

```
1169 \cs_new_protected:Npn \@@_begin_of_row:
1170   {
1171     \int_gincr:N \c@iRow
1172     \dim_gset_eq:NN \g_@@_dp_ante_last_row_dim \g_@@_dp_last_row_dim
```

```

1173 \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
1174 \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1175 \pgfpicture
1176 \pgfrememberpicturepositiononpagetrue
1177 \pgfcoordinate
1178   { \@@_env: - row - \int_use:N \c@iRow - base }
1179   { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
1180 \str_if_empty:NF \l_@@_name_str
1181   {
1182     \pgfnodealias
1183     { \l_@@_name_str - row - \int_use:N \c@iRow - base }
1184     { \@@_env: - row - \int_use:N \c@iRow - base }
1185   }
1186 \endpgfpicture
1187 }

```

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.

```

1188 \cs_new_protected:Npn \@@_update_for_first_and_last_row:
1189   {
1190     \int_compare:nNnTF \c@iRow = 0
1191     {
1192       \dim_gset:Nn \g_@@_dp_row_zero_dim
1193       { \dim_max:nn \g_@@_dp_row_zero_dim { \box_dp:N \l_@@_cell_box } }
1194       \dim_gset:Nn \g_@@_ht_row_zero_dim
1195       { \dim_max:nn \g_@@_ht_row_zero_dim { \box_ht:N \l_@@_cell_box } }
1196     }
1197     {
1198       \int_compare:nNnT \c@iRow = 1
1199       {
1200         \dim_gset:Nn \g_@@_ht_row_one_dim
1201         { \dim_max:nn \g_@@_ht_row_one_dim { \box_ht:N \l_@@_cell_box } }
1202       }
1203     }
1204   }
1205 \cs_new_protected:Npn \@@_rotate_cell_box:
1206   {
1207     \box_rotate:Nn \l_@@_cell_box { 90 }
1208     \bool_if:NTF \g_@@_rotate_c_bool
1209     {
1210       \hbox_set:Nn \l_@@_cell_box
1211       {
1212         \c_math_toggle_token
1213         \vcenter { \box_use:N \l_@@_cell_box }
1214         \c_math_toggle_token
1215       }
1216     }
1217     {
1218       \int_compare:nNnT \c@iRow = \l_@@_last_row_int
1219       {
1220         \vbox_set_top:Nn \l_@@_cell_box
1221         {
1222           \vbox_to_zero:n { }
1223           \skip_vertical:n { - \box_ht:N \@arstrutbox + 0.8 ex }
1224           \box_use:N \l_@@_cell_box
1225         }
1226       }
1227     }
1228     \bool_gset_false:N \g_@@_rotate_bool

```

```

1229   \bool_gset_false:N \g_@@_rotate_c_bool
1230   }
1231 \cs_new_protected:Npn \@@_adjust_size_box:
1232   {
1233   \dim_compare:nNnT \g_@@_blocks_wd_dim > \c_zero_dim
1234     {
1235     \box_set_wd:Nn \l_@@_cell_box
1236       { \dim_max:nn { \box_wd:N \l_@@_cell_box } \g_@@_blocks_wd_dim }
1237     \dim_gzero:N \g_@@_blocks_wd_dim
1238     }
1239   \dim_compare:nNnT \g_@@_blocks_dp_dim > \c_zero_dim
1240     {
1241     \box_set_dp:Nn \l_@@_cell_box
1242       { \dim_max:nn { \box_dp:N \l_@@_cell_box } \g_@@_blocks_dp_dim }
1243     \dim_gzero:N \g_@@_blocks_dp_dim
1244     }
1245   \dim_compare:nNnT \g_@@_blocks_ht_dim > \c_zero_dim
1246     {
1247     \box_set_ht:Nn \l_@@_cell_box
1248       { \dim_max:nn { \box_ht:N \l_@@_cell_box } \g_@@_blocks_ht_dim }
1249     \dim_gzero:N \g_@@_blocks_ht_dim
1250     }
1251   }
1252 \cs_new_protected:Npn \@@_cell_end:
1253   {
1254   \@@_math_toggle_token:
1255   \hbox_set_end:
1256   \@@_cell_end_i:
1257   }
1258 \cs_new_protected:Npn \@@_cell_end_i:
1259   {

```

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.

```

1260   \g_@@_cell_after_hook_tl
1261   \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
1262   \@@_adjust_size_box:
1263   \box_set_ht:Nn \l_@@_cell_box
1264     { \box_ht:N \l_@@_cell_box + \l_@@_cell_space_top_limit_dim }
1265   \box_set_dp:Nn \l_@@_cell_box
1266     { \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”).

```

1267   \dim_gset:Nn \g_@@_max_cell_width_dim
1268     { \dim_max:nn \g_@@_max_cell_width_dim { \box_wd:N \l_@@_cell_box } }

```

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

```

1269   \@@_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.

```

1270 \bool_if:NTF \g_@@_empty_cell_bool
1271   { \box_use_drop:N \l_@@_cell_box }
1272   {
1273     \bool_lazy_or:nnTF
1274       \g_@@_not_empty_cell_bool
1275       { \dim_compare_p:nNn { \box_wd:N \l_@@_cell_box } > \c_zero_dim }
1276       \@@_node_for_cell:
1277       { \box_use_drop:N \l_@@_cell_box }
1278   }
1279 \int_gset:Nn \g_@@_col_total_int { \int_max:nn \g_@@_col_total_int \c_jCol }
1280 \bool_gset_false:N \g_@@_empty_cell_bool
1281 \bool_gset_false:N \g_@@_not_empty_cell_bool
1282 }

```

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`).

```

1283 \cs_new_protected:Npn \@@_cell_end_for_w_s:
1284   {
1285     \@@_math_toggle_token:
1286     \hbox_set_end:
1287     \bool_if:NF \g_@@_rotate_bool
1288     {
1289       \hbox_set:Nn \l_@@_cell_box
1290       {
1291         \makebox [ \l_@@_col_width_dim ] [ s ]
1292         { \hbox_unpack_drop:N \l_@@_cell_box }
1293       }
1294     }
1295     \@@_cell_end_i:
1296   }

```

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

```

1297 \pgfset
1298   {
1299     nicematrix / cell-node /.style =
1300     {
1301       inner~sep = \c_zero_dim ,
1302       minimum~width = \c_zero_dim
1303     }
1304   }
1305 \cs_new_protected:Npn \@@_node_for_cell:
1306   {
1307     \pgfpicture
1308     \pgfsetbaseline \c_zero_dim
1309     \pgfrememberpicturepositiononpagetrue
1310     \pgfset { nicematrix / cell-node }
1311     \pgfnode
1312       { rectangle }
1313       { base }
1314       {

```

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).

```

1315     \set@color
1316     \box_use_drop:N \l_@@_cell_box
1317   }
1318   { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1319   { \l_@@_pgf_node_code_tl }
1320   \str_if_empty:NF \l_@@_name_str
1321   {
1322     \pgfnodealias
1323     { \l_@@_name_str - \int_use:N \c@iRow - \int_use:N \c@jCol }
1324     { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1325   }
1326   \endpgfpicture
1327 }

```

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.

```

1328 \cs_new_protected:Npn \@@_patch_node_for_cell:n #1
1329 {
1330   \cs_new_protected:Npn \@@_patch_node_for_cell:
1331   {
1332     \hbox_set:Nn \l_@@_cell_box
1333     {
1334       \box_move_up:nn { \box_ht:N \l_@@_cell_box}
1335       \hbox_overlap_left:n
1336       {
1337         \pgfsys@markposition
1338         { \@@_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.

```

1339         #1
1340       }
1341       \box_use:N \l_@@_cell_box
1342       \box_move_down:nn { \box_dp:N \l_@@_cell_box }
1343       \hbox_overlap_left:n
1344       {
1345         \pgfsys@markposition
1346         { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - SE }
1347         #1
1348       }
1349     }
1350   }
1351 }

```

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

```

1352 \bool_lazy_or:nnTF \sys_if_engine_xetex_p: \sys_if_output_dvi_p:
1353 {
1354   \@@_patch_node_for_cell:n
1355   { \skip_horizontal:n { 0.5 \box_wd:N \l_@@_cell_box } }
1356 }
1357 { \@@_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}

```

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

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).

```

1358 \cs_new_protected:Npn \@@_instruction_of_type:nnn #1 #2 #3
1359   {
1360     \bool_if:nTF { #1 } \tl_gput_left:cx \tl_gput_right:cx
1361     { \g_@@_ #2 _ lines _ tl }
1362     {
1363       \use:c { @@ _ draw _ #2 : nnn }
1364       { \int_use:N \c@iRow }
1365       { \int_use:N \c@jCol }
1366       { \exp_not:n { #3 } }
1367     }
1368   }
1369 \cs_new_protected:Npn \@@_array:n
1370   {
1371     % modified 05-08-23
1372     \dim_set:Nn \col@sep
1373     { \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
1374     \dim_compare:nNnTF \l_@@_tabular_width_dim = \c_zero_dim
1375     { \cs_set_nopar:Npn \@halignto { } }
1376     { \cs_set_nopar:Npx \@halignto { to \dim_use:N \l_@@_tabular_width_dim } }

```

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

```

1377   \@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.

```

1378   [ \str_if_eq:VnTF \l_@@_baseline_tl c c t ]
1379   }
1380 \cs_generate_variant:Nn \@@_array:n { V }

```

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.

```

1381 \cs_set_eq:NN \@@_old_ialign: \ialign

```

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

```

1382 \cs_new_protected:Npn \@@_create_row_node:
1383   {
1384     \int_compare:nNnT \c@iRow > \g_@@_last_row_node_int
1385     {
1386       \int_gset_eq:NN \g_@@_last_row_node_int \c@iRow
1387       \@@_create_row_node_i:
1388     }
1389   }
1390 \cs_new_protected:Npn \@@_create_row_node_i:
1391   {

```

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

```

1392   \hbox
1393   {
1394     \bool_if:NT \l_@@_code_before_bool

```



```

1395     {
1396         \vtop
1397         {
1398             \skip_vertical:N 0.5\arrayrulewidth
1399             \pgfsys@markposition
1400             { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1401             \skip_vertical:N -0.5\arrayrulewidth
1402         }
1403     }
1404     \pgfpicture
1405     \pgfrememberpicturepositiononpagetrue
1406     \pgfcoordinate { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1407     { \pgfpoint \c_zero_dim { - 0.5 \arrayrulewidth } }
1408     \str_if_empty:NF \l_@@_name_str
1409     {
1410         \pgfnodealias
1411         { \l_@@_name_str - row - \int_eval:n { \c@iRow + 1 } }
1412         { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1413     }
1414     \endpgfpicture
1415 }
1416 }

```

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

```

1417 \cs_new:Npn \@@_everycr: { \noalign { \@@_everycr_i: } }
1418 \cs_new_protected:Npn \@@_everycr_i:
1419 {
1420     \int_gzero:N \c@jCol
1421     \bool_gset_false:N \g_@@_after_col_zero_bool
1422     \bool_if:NF \g_@@_row_of_col_done_bool
1423     {
1424         \@@_create_row_node:

```

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

```

1425     \tl_if_empty:NF \l_@@_hlines_clist
1426     {
1427         \tl_if_eq:NnF \l_@@_hlines_clist { all }
1428         {
1429             \exp_args:NNx
1430             \clist_if_in:NnT
1431             \l_@@_hlines_clist
1432             { \int_eval:n { \c@iRow + 1 } }
1433         }
1434     }

```

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.

```

1435         \int_compare:nNnT \c@iRow > { -1 }
1436         {
1437             \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@`.

```

1438             { \hrule height \arrayrulewidth width \c_zero_dim }
1439         }
1440     }
1441 }
1442 }
1443 }

```

The command `\@@_newcolumntype` is the command `\newcolumntype` of `array` without the warnings for redefinitions of columns types (we will use it to redefine the columns types `w` and `W`).

```

1444 \cs_set_protected:Npn \@@_newcolumntype #1
1445   {
1446     \cs_set:cpn { NC @ find @ #1 } ##1 #1 { \NC@ { ##1 } }
1447     \peek_meaning:NTF [
1448       { \newcol@ #1 }
1449       { \newcol@ #1 [ 0 ] }
1450   }

```

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

```

1451 \cs_set_protected:Npn \@@_renew_dots:
1452   {
1453     \cs_set_eq:NN \ldots \@@_Ldots
1454     \cs_set_eq:NN \cdots \@@_Cdots
1455     \cs_set_eq:NN \vdots \@@_Vdots
1456     \cs_set_eq:NN \ddots \@@_Ddots
1457     \cs_set_eq:NN \iddots \@@_Iddots
1458     \cs_set_eq:NN \dots \@@_Ldots
1459     \cs_set_eq:NN \hdotsfor \@@_Hdotsfor:
1460   }

```

When the key `color-inside` is used, the following code will be executed.

```

1461 \cs_new_protected:Npn \@@_colortbl_like:
1462   {
1463     \cs_set_eq:NN \cellcolor \@@_cellcolor_tabular
1464     \cs_set_eq:NN \rowcolor \@@_rowcolor_tabular
1465     \cs_set_eq:NN \columncolor \@@_columncolor_preamble
1466     \cs_set_eq:NN \rowcolors \@@_rowcolors_tabular
1467     \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors_tabular
1468   }

```

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

```

1469 \cs_new_protected:Npn \@@_pre_array_ii:
1470   {

```

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

```

1471   \int_gzero:N \g_@@_total_X_weight_int
1472   \@@_expand_clist:N \l_@@_hlines_clist
1473   \@@_expand_clist:N \l_@@_vlines_clist

```

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.

```

1474   \IfPackageLoadedTF { booktabs }
1475     { \tl_put_left:Nn \@BTnormal \@@_create_row_node_i: }
1476     { }
1477   \box_clear_new:N \l_@@_cell_box
1478   \normalbaselines

```

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

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}`).

```

1479   \bool_if:NT \l_@@_small_bool
1480   {
1481     \cs_set_nopar:Npn \arraystretch { 0.47 }
1482     \dim_set:Nn \arraycolsep { 1.45 pt }
1483   }

1484   \bool_if:NT \g_@@_recreate_cell_nodes_bool
1485   {
1486     \tl_put_right:Nn \@@_begin_of_row:
1487     {
1488       \pgfsys@markposition
1489       { \@@_env: - row - \int_use:N \c@iRow - base }
1490     }
1491   }

```

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`.

```

1492   \cs_set_nopar:Npn \ialign
1493   {
1494     \IfPackageLoadedTF { colortbl }
1495     {
1496       \CT@everycr
1497       {
1498         \noalign { \cs_gset_eq:NN \CT@row@color \prg_do_nothing: }
1499         \@@_everycr:
1500       }
1501     }
1502     { \everycr { \@@_everycr: } }
1503     \tabskip = \c_zero_skip

```

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`.

```

1504     \dim_gzero_new:N \g_@@_dp_row_zero_dim
1505     \dim_gset:Nn \g_@@_dp_row_zero_dim { \box_dp:N \@arstrutbox }
1506     \dim_gzero_new:N \g_@@_ht_row_zero_dim
1507     \dim_gset:Nn \g_@@_ht_row_zero_dim { \box_ht:N \@arstrutbox }
1508     \dim_gzero_new:N \g_@@_ht_row_one_dim
1509     \dim_gset:Nn \g_@@_ht_row_one_dim { \box_ht:N \@arstrutbox }
1510     \dim_gzero_new:N \g_@@_dp_ante_last_row_dim
1511     \dim_gzero_new:N \g_@@_ht_last_row_dim
1512     \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1513     \dim_gzero_new:N \g_@@_dp_last_row_dim
1514     \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`.

```

1515     \cs_set_eq:NN \ialign \@@_old_ialign:
1516     \halign
1517   }

```

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

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).

```

1518 \cs_set_eq:NN \@@_old_ldots \ldots
1519 \cs_set_eq:NN \@@_old_cdots \cdots
1520 \cs_set_eq:NN \@@_old_vdots \vdots
1521 \cs_set_eq:NN \@@_old_ddots \ddots
1522 \cs_set_eq:NN \@@_old_iddots \iddots
1523 \bool_if:NTF \l_@@_standard_cline_bool
1524 { \cs_set_eq:NN \cline \@@_standard_cline }
1525 { \cs_set_eq:NN \cline \@@_cline }
1526 \cs_set_eq:NN \Ldots \@@_Ldots
1527 \cs_set_eq:NN \Cdots \@@_Cdots
1528 \cs_set_eq:NN \Vdots \@@_Vdots
1529 \cs_set_eq:NN \Ddots \@@_Ddots
1530 \cs_set_eq:NN \Iddots \@@_Iddots
1531 \cs_set_eq:NN \Hline \@@_Hline:
1532 \cs_set_eq:NN \Hspace \@@_Hspace:
1533 \cs_set_eq:NN \Hdotsfor \@@_Hdotsfor:
1534 \cs_set_eq:NN \Vdotsfor \@@_Vdotsfor:
1535 \cs_set_eq:NN \Block \@@_Block:
1536 \cs_set_eq:NN \rotate \@@_rotate:
1537 \cs_set_eq:NN \OnlyMainNiceMatrix \@@_OnlyMainNiceMatrix:n
1538 \cs_set_eq:NN \dotfill \@@_dotfill:
1539 \cs_set_eq:NN \CodeAfter \@@_CodeAfter:
1540 \cs_set_eq:NN \diagbox \@@_diagbox:nn
1541 \cs_set_eq:NN \NotEmpty \@@_NotEmpty:
1542 \cs_set_eq:NN \RowStyle \@@_RowStyle:n
1543 \seq_map_inline:Nn \l_@@_custom_line_commands_seq
1544 { \cs_set_eq:cc { ##1 } { nicematrix - ##1 } }
1545 \bool_if:NT \l_@@_color_inside_bool \@@_colortbl_like:
1546 \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.

```

1547 \cs_set_eq:NN \multicolumn \@@_multicolumn:nnn
1548 \hook_gput_code:nnn { env / tabular / begin } { . }
1549 { \cs_set_eq:NN \multicolumn \@@_old_multicolumn }

```

If there is one or several commands `\tabularnote` in the caption specified by the key `caption` and if that caption has to be composed above the tabular, we have now that information because it has been written in the aux file at a previous run. We use that information to start counting the tabular notes in the main array at the right value (we remember that the caption will be composed *after* the array!).

```

1550 \tl_if_exist:NT \l_@@_note_in_caption_tl
1551 {
1552   \tl_if_empty:NF \l_@@_note_in_caption_tl
1553   {
1554     \int_gset_eq:NN \g_@@_notes_caption_int \l_@@_note_in_caption_tl
1555     \int_gset:Nn \c@tabularnote { \l_@@_note_in_caption_tl }
1556   }
1557 }

```

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).

```

1558 \seq_gclear:N \g_@@_multicolumn_cells_seq
1559 \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).

```

1560 \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`).

```
1561 \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.

```
1562 \int_gzero_new:N \g_@@_col_total_int
1563 \cs_set_eq:NN \@ifnextchar \new@ifnextchar
1564 \@@_renew_NC@rewrite@S:
1565 \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.

```
1566 \tl_gclear_new:N \g_@@_Cdots_lines_tl
1567 \tl_gclear_new:N \g_@@_Ldots_lines_tl
1568 \tl_gclear_new:N \g_@@_Vdots_lines_tl
1569 \tl_gclear_new:N \g_@@_Ddots_lines_tl
1570 \tl_gclear_new:N \g_@@_Iddots_lines_tl
1571 \tl_gclear_new:N \g_@@_HVdotsfor_lines_tl

1572 \tl_gclear:N \g_nicematrix_code_before_tl
1573 \tl_gclear:N \g_@@_pre_code_before_tl
1574 }
```

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

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

```
1575 \cs_new_protected:Npn \@@_pre_array:
1576 {
1577   \cs_if_exist:NT \theiRow { \int_set_eq:NN \l_@@_old_iRow_int \c@iRow }
1578   \int_gzero_new:N \c@iRow
1579   \cs_if_exist:NT \thejCol { \int_set_eq:NN \l_@@_old_jCol_int \c@jCol }
1580   \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).

```
1581 \int_compare:nNnT \l_@@_last_row_int = { -1 }
1582 {
1583   \bool_set_true:N \l_@@_last_row_without_value_bool
1584   \bool_if:NT \g_@@_aux_found_bool
1585     { \int_set:Nn \l_@@_last_row_int { \seq_item:Nn \g_@@_size_seq 3 } }
1586 }
1587 \int_compare:nNnT \l_@@_last_col_int = { -1 }
1588 {
1589   \bool_if:NT \g_@@_aux_found_bool
1590     { \int_set:Nn \l_@@_last_col_int { \seq_item:Nn \g_@@_size_seq 6 } }
1591 }
```

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”.

```
1592 \int_compare:nNnT \l_@@_last_row_int > { -2 }
1593 {
1594   \tl_put_right:Nn \@@_update_for_first_and_last_row:
1595     {
```

```

1596         \dim_gset:Nn \g_@@_ht_last_row_dim
1597         { \dim_max:nn \g_@@_ht_last_row_dim { \box_ht:N \l_@@_cell_box } }
1598         \dim_gset:Nn \g_@@_dp_last_row_dim
1599         { \dim_max:nn \g_@@_dp_last_row_dim { \box_dp:N \l_@@_cell_box } }
1600     }
1601 }

```

```

1602 \seq_gclear:N \g_@@_cols_vlism_seq
1603 \seq_gclear:N \g_@@_submatrix_seq

```

Now the `\CodeBefore`.

```

1604 \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.

```

1605 \seq_gclear:N \g_@@_pos_of_blocks_seq

```

Idem for other sequences written on the aux file.

```

1606 \seq_gclear_new:N \g_@@_multicolumn_cells_seq
1607 \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.

```

1608 \int_gset:Nn \g_@@_last_row_node_int { -2 }

```

The value `-2` is important.

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

```

1609 \@@_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.

```

1610 \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]`).

```

1611 \dim_zero_new:N \l_@@_left_delim_dim
1612 \dim_zero_new:N \l_@@_right_delim_dim
1613 \bool_if:NTF \g_@@_delims_bool
1614 {

```

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

```

1615 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_left_delim_tl $ }
1616 \dim_set:Nn \l_@@_left_delim_dim { \box_wd:N \l_tmpa_box }
1617 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_right_delim_tl $ }
1618 \dim_set:Nn \l_@@_right_delim_dim { \box_wd:N \l_tmpa_box }
1619 }
1620 {
1621 % modified 05-08-23
1622 \dim_gset:Nn \l_@@_left_delim_dim
1623 { 2 \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
1624 \dim_gset_eq:NN \l_@@_right_delim_dim \l_@@_left_delim_dim
1625 }

```

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).

```

1626   \hbox_set:Nw \l_@@_the_array_box
1627   \skip_horizontal:N \l_@@_left_margin_dim
1628   \skip_horizontal:N \l_@@_extra_left_margin_dim
1629   \c_math_toggle_token
1630   \bool_if:NTF \l_@@_light_syntax_bool
1631     { \use:c { @@-light-syntax } }
1632     { \use:c { @@-normal-syntax } }
1633   }

```

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

```

1634 \cs_new_protected_nopar:Npn \@@_CodeBefore_Body:w #1 \Body
1635   {
1636     \tl_gput_left:Nn \g_@@_pre_code_before_tl { #1 }
1637     \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...

```

1638   \@@_pre_array:
1639 }

```

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).

```

1640 \cs_new_protected:Npn \@@_pre_code_before:
1641   {

```

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).

```

1642   \int_set:Nn \c@iRow { \seq_item:Nn \g_@@_size_seq 2 }
1643   \int_set:Nn \c@jCol { \seq_item:Nn \g_@@_size_seq 5 }
1644   \int_set_eq:NN \g_@@_row_total_int { \seq_item:Nn \g_@@_size_seq 3 }
1645   \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.

```

1646   \pgfsys@markposition { \@@_env: - position }
1647   \pgfsys@getposition { \@@_env: - position } \@@_picture_position:
1648   \pgfpicture
1649   \pgf@relevantforpicturesizefalse

```

First, the recreation of the row nodes.

```

1650   \int_step_inline:nnn \l_@@_first_row_int { \g_@@_row_total_int + 1 }
1651   {
1652     \pgfsys@getposition { \@@_env: - row - ##1 } \@@_node_position:
1653     \pgfcoordinate { \@@_env: - row - ##1 }
1654     { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1655   }

```

Now, the recreation of the col nodes.

```

1656 \int_step_inline:nnn \l_@@_first_col_int { \g_@@_col_total_int + 1 }
1657 {
1658   \pgfsys@getposition { \@@_env: - col - ##1 } \@@_node_position:
1659   \pgfcoordinate { \@@_env: - col - ##1 }
1660   { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1661 }

```

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

```

1662 \@@_create_diag_nodes:

```

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

```

1663 \bool_if:NT \g_@@_recreate_cell_nodes_bool \@@_recreate_cell_nodes:
1664 \endpgfpicture

```

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

```

1665 \@@_create_blocks_nodes:
1666 \IfPackageLoadedTF { tikz }
1667 {
1668   \tikzset
1669   {
1670     every~picture / .style =
1671     { overlay , name~prefix = \@@_env: - }
1672   }
1673 }
1674 { }
1675 \cs_set_eq:NN \cellcolor \@@_cellcolor
1676 \cs_set_eq:NN \rectanglecolor \@@_rectanglecolor
1677 \cs_set_eq:NN \roundedrectanglecolor \@@_roundedrectanglecolor
1678 \cs_set_eq:NN \rowcolor \@@_rowcolor
1679 \cs_set_eq:NN \rowcolors \@@_rowcolors
1680 \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors
1681 \cs_set_eq:NN \arraycolor \@@_arraycolor
1682 \cs_set_eq:NN \columncolor \@@_columncolor
1683 \cs_set_eq:NN \chessboardcolors \@@_chessboardcolors
1684 \cs_set_eq:NN \SubMatrix \@@_SubMatrix_in_code_before
1685 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
1686 }

```

```

1687 \cs_new_protected:Npn \@@_exec_code_before:
1688 {
1689   \seq_gclear_new:N \g_@@_colors_seq
1690   \bool_gset_false:N \g_@@_recreate_cell_nodes_bool
1691   \group_begin:

```

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

```

1692 \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`).

```

1693 \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
1694 {
1695   \@@_rescan_for_spanish:N \g_@@_pre_code_before_tl
1696   \@@_rescan_for_spanish:N \l_@@_code_before_tl
1697 }

```


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`.

```
1698 \exp_last_unbraced:NV \@@_CodeBefore_keys:
1699 \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.

```
1700 \@@_actually_color:
1701 \l_@@_code_before_tl
1702 \q_stop
1703 \bool_if:NT \l_@@_tabular_bool \c_math_toggle_token
1704 \group_end:
1705 \bool_if:NT \g_@@_recreate_cell_nodes_bool
1706 { \tl_put_left:Nn \@@_node_for_cell: \@@_patch_node_for_cell: }
1707 }
```

```
1708 \keys_define:nn { NiceMatrix / CodeBefore }
1709 {
1710 create-cell-nodes .bool_gset:N = \g_@@_recreate_cell_nodes_bool ,
1711 create-cell-nodes .default:n = true ,
1712 sub-matrix .code:n = \keys_set:nn { NiceMatrix / sub-matrix } { #1 } ,
1713 sub-matrix .value_required:n = true ,
1714 delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1715 delimiters / color .value_required:n = true ,
1716 unknown .code:n = \@@_error:n { Unknown-key-for-CodeBefore }
1717 }
1718 \NewDocumentCommand \@@_CodeBefore_keys: { 0 { } }
1719 {
1720 \keys_set:nn { NiceMatrix / CodeBefore } { #1 }
1721 \@@_CodeBefore:w
1722 }
```

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.

```
1723 \cs_new_protected:Npn \@@_CodeBefore:w #1 \q_stop
1724 {
1725 \bool_if:NT \g_@@_aux_found_bool
1726 {
1727 \@@_pre_code_before:
1728 #1
1729 }
1730 }
```

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.

```
1731 \cs_new_protected:Npn \@@_recreate_cell_nodes:
1732 {
1733 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
1734 {
1735 \pgfsys@getposition { \@@_env: - ##1 - base } \@@_node_position:
1736 \pgfcoordinate { \@@_env: - row - ##1 - base }
1737 { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1738 \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
1739 {
```

```

1740     \cs_if_exist:cT
1741     { pgf @ sys @ pdf @ mark @ pos @ \@@_env: - ##1 - #####1 - NW }
1742     {
1743         \pgfsys@getposition
1744         { \@@_env: - ##1 - #####1 - NW }
1745         \@@_node_position:
1746         \pgfsys@getposition
1747         { \@@_env: - ##1 - #####1 - SE }
1748         \@@_node_position_i:
1749         \@@_pgf_rect_node:nnn
1750         { \@@_env: - ##1 - #####1 }
1751         { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1752         { \pgfpointdiff \@@_picture_position: \@@_node_position_i: }
1753     }
1754 }
1755 }
1756 \int_step_inline:nn \c@iRow
1757 {
1758     \pgfnodealias
1759     { \@@_env: - ##1 - last }
1760     { \@@_env: - ##1 - \int_use:N \c@jCol }
1761 }
1762 \int_step_inline:nn \c@jCol
1763 {
1764     \pgfnodealias
1765     { \@@_env: - last - ##1 }
1766     { \@@_env: - \int_use:N \c@iRow - ##1 }
1767 }
1768 \@@_create_extra_nodes:
1769 }

1770 \cs_new_protected:Npn \@@_create_blocks_nodes:
1771 {
1772     \pgfpicture
1773     \pgf@relevantforpicturesizefalse
1774     \pgfrememberpicturepositiononpagetrue
1775     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
1776     { \@@_create_one_block_node:nnnnn ##1 }
1777     \endpgfpicture
1778 }

```

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

```

1779 \cs_new_protected:Npn \@@_create_one_block_node:nnnnn #1 #2 #3 #4 #5
1780 {
1781     \tl_if_empty:nF { #5 }
1782     {
1783         \@@_qpoint:n { col - #2 }
1784         \dim_set_eq:NN \l_tmpa_dim \pgf@x
1785         \@@_qpoint:n { #1 }
1786         \dim_set_eq:NN \l_tmpb_dim \pgf@y
1787         \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
1788         \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
1789         \@@_qpoint:n { \int_eval:n { #3 + 1 } }
1790         \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
1791         \@@_pgf_rect_node:nnnnn
1792         { \@@_env: - #5 }
1793         { \dim_use:N \l_tmpa_dim }
1794         { \dim_use:N \l_tmpb_dim }

```

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).

```

1795     { \dim_use:N \l_@@_tmpc_dim }
1796     { \dim_use:N \l_@@_tmpd_dim }
1797   }
1798 }

1799 \cs_new_protected:Npn \@@_patch_for_revtext:
1800 {
1801   \cs_set_eq:NN \@addamp \@addamp@LaTeX
1802   \cs_set_eq:NN \insert@column \insert@column@array
1803   \cs_set_eq:NN \@classx \@classx@array
1804   \cs_set_eq:NN \@xarraycr \@xarraycr@array
1805   \cs_set_eq:NN \@arraycr \@arraycr@array
1806   \cs_set_eq:NN \@xargarraycr \@xargarraycr@array
1807   \cs_set_eq:NN \array \array@array
1808   \cs_set_eq:NN \@array \@array@array
1809   \cs_set_eq:NN \@tabular \@tabular@array
1810   \cs_set_eq:NN \@mkpream \@mkpream@array
1811   \cs_set_eq:NN \endarray \endarray@array
1812   \cs_set:Npn \@tabarray { \@ifnextchar [ { \@array } { \@array [ c ] } }
1813   \cs_set:Npn \endtabular { \endarray $\egroup} % $
1814 }

```

11 The environment `{NiceArrayWithDelims}`

```

1815 \NewDocumentEnvironment { NiceArrayWithDelims }
1816 { m m 0 { } m ! 0 { } t \CodeBefore }
1817 {
1818   \bool_if:NT \c_@@_revtext_bool \@@_patch_for_revtext:
1819   \@@_provide_pgfsyspdfmark:
1820   \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 exponent to a matrix in a mathematical formula.

```

1821   \bgroup

1822   \tl_gset:Nn \g_@@_left_delim_tl { #1 }
1823   \tl_gset:Nn \g_@@_right_delim_tl { #2 }
1824   \tl_gset:Nn \g_@@_preamble_tl { #4 }

1825   \int_zero:N \g_@@_block_box_int
1826   \dim_zero:N \g_@@_width_last_col_dim
1827   \dim_zero:N \g_@@_width_first_col_dim
1828   \bool_gset_false:N \g_@@_row_of_col_done_bool
1829   \str_if_empty:NT \g_@@_name_env_str
1830     { \str_gset:Nn \g_@@_name_env_str { NiceArrayWithDelims } }
1831   \bool_if:NTF \l_@@_tabular_bool
1832     \mode_leave_vertical:
1833     \@@_test_if_math_mode:
1834   \bool_if:NT \l_@@_in_env_bool { \@@_fatal:n { Yet~in~env } }
1835   \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

⁷e.g. `\color[rgb]{0.5,0.5,0}`

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.

```
1836 \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.

```
1837 \cs_if_exist:NT \tikz@library@external@loaded
1838 {
1839   \tikzexternaldisable
1840   \cs_if_exist:NT \ifstandalone
1841   { \tikzset { external / optimize = false } }
1842 }
```

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

```
1843 \int_gincr:N \g_@@_env_int
1844 \bool_if:NF \l_@@_block_auto_columns_width_bool
1845 { \dim_gzero_new:N \g_@@_max_cell_width_dim }
```

The sequence `\g_@@_blocks_seq` will contain the characteristics 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`).

```
1846 \seq_gclear:N \g_@@_blocks_seq
1847 \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`.

```
1848 \seq_gclear:N \g_@@_pos_of_stroken_blocks_seq
1849 \seq_gclear:N \g_@@_pos_of_xdots_seq
1850 \tl_gclear_new:N \g_@@_code_before_tl
1851 \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.

```
1852 \tl_if_exist:cTF { c_@@ _ \int_use:N \g_@@_env_int _ tl }
1853 {
1854   \bool_gset_true:N \g_@@_aux_found_bool
1855   \use:c { c_@@ _ \int_use:N \g_@@_env_int _ tl }
1856 }
1857 { \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.

```
1858 \tl_gclear:N \g_@@_aux_tl
1859 \tl_if_empty:NF \g_@@_code_before_tl
1860 {
1861   \bool_set_true:N \l_@@_code_before_bool
1862   \tl_put_right:NV \l_@@_code_before_tl \g_@@_code_before_tl
1863 }
1864 \tl_if_empty:NF \g_@@_pre_code_before_tl
1865 { \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`.

```
1866 \bool_if:NTF \g_@@_delims_bool
1867 { \keys_set:nn { NiceMatrix / pNiceArray } }
1868 { \keys_set:nn { NiceMatrix / NiceArray } }
1869 { #3 , #5 }

1870 \@@_set_CT@arc@:V \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:`.

```
1871   \IfBooleanTF { #6 } \@@_CodeBefore_Body:w \@@_pre_array:
1872   }
```

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

```
1873   {
1874     \bool_if:NTF \l_@@_light_syntax_bool
1875     { \use:c { end @@-light-syntax } }
1876     { \use:c { end @@-normal-syntax } }
1877     \c_math_toggle_token
1878     \skip_horizontal:N \l_@@_right_margin_dim
1879     \skip_horizontal:N \l_@@_extra_right_margin_dim
1880     \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.

```
1881   \bool_if:NT \l_@@_width_used_bool
1882   {
1883     \int_compare:nNnT \g_@@_total_X_weight_int = 0
1884     { \@@_error_or_warning:n { width-without~X-columns } }
1885   }
```

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`.

```
1886   \int_compare:nNnT \g_@@_total_X_weight_int > 0
1887   {
1888     \tl_gput_right:Nx \g_@@_aux_tl
1889     {
1890       \bool_set_true:N \l_@@_X_columns_aux_bool
1891       \dim_set:Nn \l_@@_X_columns_dim
1892       {
1893         \dim_compare:nNnTF
1894         {
1895           \dim_abs:n
1896           { \l_@@_width_dim - \box_wd:N \l_@@_the_array_box }
1897         }
1898         <
1899         { 0.001 pt }
1900         { \dim_use:N \l_@@_X_columns_dim }
1901         {
1902           \dim_eval:n
1903           {
1904             ( \l_@@_width_dim - \box_wd:N \l_@@_the_array_box )
1905             / \int_use:N \g_@@_total_X_weight_int
1906             + \l_@@_X_columns_dim
1907           }
1908         }
1909       }
1910     }
1911   }
```

If the user has used the key `last-row` with a value, we control that the given value is correct (since we have just constructed the array, we know the actual number of rows of the array).

```
1912   \int_compare:nNnT \l_@@_last_row_int > { -2 }
1913   {
1914     \bool_if:NF \l_@@_last_row_without_value_bool
1915     {
1916       \int_compare:nNnF \l_@@_last_row_int = \c_iRow
1917     }
```

```

1918         \@@_error:n { Wrong-last-row }
1919         \int_gset_eq:NN \l_@@_last_row_int \c@iRow
1920     }
1921 }
1922 }

```

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

```

1923     \int_gset_eq:NN \c@jCol \g_@@_col_total_int
1924     \bool_if:nTF \g_@@_last_col_found_bool
1925     { \int_gdecr:N \c@jCol }
1926     {
1927         \int_compare:nNnT \l_@@_last_col_int > { -1 }
1928         { \@@_error:n { last-col-not-used } }
1929     }

```

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

```

1930     \int_gset_eq:NN \g_@@_row_total_int \c@iRow
1931     \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. 86).

```

1932     \int_compare:nNnT \l_@@_first_col_int = 0
1933     {
1934         % \skip_horizontal:N \col@sep % 05-08-23
1935         \skip_horizontal:N \g_@@_width_first_col_dim
1936     }

```

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

```

1937     \bool_if:nTF { ! \g_@@_delims_bool }
1938     {
1939         \str_case:VnF \l_@@_baseline_tl
1940         {
1941             b \@@_use_arraybox_with_notes_b:
1942             c \@@_use_arraybox_with_notes_c:
1943         }
1944         \@@_use_arraybox_with_notes:
1945     }

```

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).

```

1946     {
1947         \int_compare:nNnTF \l_@@_first_row_int = 0
1948         {
1949             \dim_set_eq:NN \l_tmpa_dim \g_@@_dp_row_zero_dim
1950             \dim_add:Nn \l_tmpa_dim \g_@@_ht_row_zero_dim
1951         }
1952         { \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

```

1953     \int_compare:nNnTF \l_@@_last_row_int > { -2 }
1954     {
1955         \dim_set_eq:NN \l_tmpb_dim \g_@@_ht_last_row_dim
1956         \dim_add:Nn \l_tmpb_dim \g_@@_dp_last_row_dim
1957     }
1958     { \dim_zero:N \l_tmpb_dim }
1959     \hbox_set:Nn \l_tmpa_box
1960     {

```

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

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

```

1961         \c_math_toggle_token
1962         \@@_color:V \l_@@_delimiters_color_tl
1963         \exp_after:wN \left \g_@@_left_delim_tl
1964         \vcenter
1965         {

```

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.

```

1966         \skip_vertical:n { -\l_tmpa_dim - \arrayrulewidth }
1967         \hbox
1968         {
1969             \bool_if:NTF \l_@@_tabular_bool
1970             { \skip_horizontal:N -\tabcolsep }
1971             { \skip_horizontal:N -\arraycolsep }
1972             \@@_use_arraybox_with_notes_c:
1973             \bool_if:NTF \l_@@_tabular_bool
1974             { \skip_horizontal:N -\tabcolsep }
1975             { \skip_horizontal:N -\arraycolsep }
1976         }

```

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

```

1977         \skip_vertical:n { -\l_tmpb_dim + \arrayrulewidth }
1978     }

```

Curiously, we have to put again the following specification of color. Otherwise, with XeLaTeX (and not with the other engines), the closing delimiter is not colored.

```

1979         \@@_color:V \l_@@_delimiters_color_tl
1980         \exp_after:wN \right \g_@@_right_delim_tl
1981         \c_math_toggle_token
1982     }

```

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.

```

1983     \bool_if:NTF \l_@@_delimiters_max_width_bool
1984     {
1985         \@@_put_box_in_flow_bis:nn
1986         \g_@@_left_delim_tl \g_@@_right_delim_tl
1987     }
1988     \@@_put_box_in_flow:
1989 }

```

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. 87).

```

1990     \bool_if:NT \g_@@_last_col_found_bool
1991     {
1992         \skip_horizontal:N \g_@@_width_last_col_dim
1993         % \skip_horizontal:N \col@sep % 2023-08-05
1994     }
1995     \bool_if:NT \l_@@_preamble_bool
1996     {
1997         \int_compare:nNnT \c@jCol < \g_@@_static_num_of_col_int
1998         { \@@_warning_gredirect_none:n { columns~not~used } }
1999     }
2000     \@@_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 exponent to a matrix in a mathematical formula.

```

2001     \egroup

```

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

```

2002     \iow_now:Nn \@mainaux { \ExplSyntaxOn }
2003     \iow_now:Nn \@mainaux { \char_set_catcode_space:n { 32 } }
2004     \iow_now:Nx \@mainaux
2005     {
2006         \tl_gset:cn { c_@@_ \int_use:N \g_@@_env_int _ tl }

```

```

2007         { \exp_not:V \g_@@_aux_tl }
2008     }
2009     \iow_now:Nn \@mainaux { \ExplSyntaxOff }

2010     \bool_if:NT \g_@@_footnote_bool \endsavenotes
2011 }

```

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

12 We construct the preamble of the array

The transformation of the preamble is an operation in several steps.¹⁰

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

```

2012 \cs_new_protected:Npn \@@_transform_preamble:
2013 {
2014     \bool_if:NT \l_@@_preamble_bool \@@_transform_preamble_i:
2015     \@@_transform_preamble_ii:
2016 }

2017 \cs_new_protected:Npn \@@_transform_preamble_i:
2018 {

```

First, we will do an “expansion” of the preamble with the tools of the package `array` itself. This “expansion” will expand all the constructions with `*` and all column types (defined by the user or by various packages using `\newcolumntype`).

Since we use the tools of `array` to do this expansion, we will have a programming which is not in the style of the L3 programming layer.

We redefine the column types `w` and `W`. We use `\@@_newcolumntype` instead of `\newcolumntype` because we don’t want warnings for column types already defined. These redefinitions are in fact *protections* of the letters `w` and `W`. We don’t want these columns type expanded because we will do the patch ourselves after. We want to be able to use the standard column types `w` and `W` in potential `{tabular}` of `array` in some cells of our array. That’s why we do those redefinitions in a TeX group.

```

2019     \group_begin:
2020     \@@_newcolumntype w [ 2 ] { \@@_w: { ##1 } { ##2 } }
2021     \@@_newcolumntype W [ 2 ] { \@@_W: { ##1 } { ##2 } }

```

If the package `varwidth` has defined the column type `V`, we protect from expansion by redefining it to `\@@_V:` (which will be caught by our system).

```

2022     \cs_if_exist:NT \NC@find@V { \@@_newcolumntype V { \@@_V: } }

```

First, we have to store our preamble in the token register `\@temptokena` (those “token registers” are *not* supported by the L3 programming layer).

```

2023     \exp_args:NV \@temptokena \g_@@_preamble_tl

```

Initialisation of a flag used by `array` to detect the end of the expansion.

```

2024     \@tempwatrue

```

The following line actually does the expansion (it’s has been copied from `array.sty`). The expanded version is still in `\@temptokena`.

```

2025     \@whilesw \if@tempswa \fi { \@tempwafalse \the \NC@list }
2026     \int_gzero:N \c@jCol
2027     \tl_gclear:N \g_@@_preamble_tl

```

¹⁰Be careful: the transformation of the preamble may also have by-side effects, for example, the boolean `\g_@@_delims_bool` will be set to `true` if we detect in the preamble a delimiter at the beginning or at the end.

`\g_tmpb_bool` will be raised if you have a `|` at the end of the preamble.

```

2028 \bool_gset_false:N \g_tmpb_bool
2029 \tl_if_eq:NnTF \l_@@_vlines_clist { all }
2030 {
2031   \tl_gset:Nn \g_@@_preamble_tl
2032   { ! { \skip_horizontal:N \arrayrulewidth } }
2033 }
2034 {
2035   \clist_if_in:NnT \l_@@_vlines_clist 1
2036   {
2037     \tl_gset:Nn \g_@@_preamble_tl
2038     { ! { \skip_horizontal:N \arrayrulewidth } }
2039   }
2040 }

```

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`).

```

2041 \seq_clear:N \g_@@_cols_vlism_seq

```

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

```

2042 \tl_gclear_new:N \g_@@_pre_cell_tl

```

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

```

2043 \int_zero:N \l_tmpa_int

```

Now, we actually patch the preamble (and it is constructed in `\g_@@_preamble_tl`).

```

2044 \exp_after:wN \@@_patch_preamble:n \the \@temptokena \q_stop
2045 \int_gset_eq:NN \g_@@_static_num_of_col_int \c@jCol

```

Remark that `\g_@@_static_num_of_col_int` will stay equal to zero in the environments without preamble since we are in a code that is executed only in the environments *with* preamble.

Now, we replace `\columncolor` by `\@@_columncolor_preamble`.

```

2046 \bool_if:NT \l_@@_color_inside_bool
2047 {
2048   \regex_replace_all:NnN
2049   \c_@@_columncolor_regex
2050   { \c { @@_columncolor_preamble } }
2051   \g_@@_preamble_tl
2052 }

```

Now, we can close the TeX group which was opened for the redefinition of the columns of type `w` and `W`.

```

2053 \group_end:
2054 }

```

Now, we have to “patch” that preamble by transforming some columns. We will insert in the TeX flow the preamble in its actual form (that is to say after the “expansion”) following by a marker `\q_stop` and we will consume these tokens constructing the (new form of the) preamble in `\g_@@_preamble_tl`. This is done recursively with the command `\@@_patch_preamble:n`. In the same time, we will count the columns with the counter `\c@jCol`.

```

2055 \cs_new_protected:Npn \@@_transform_preamble_ii:
2056 {
2057 %
2058 % \medskip
2059 % If there was delimiters at the beginning or at the end of the preamble, the
2060 % environment {\NiceArray} is transformed into an environment {\xNiceMatrix}.
2061 % \begin{macrocode}
2062 \bool_lazy_or:nnT
2063 { ! \str_if_eq_p:Vn \g_@@_left_delim_tl { . } }
2064 { ! \str_if_eq_p:Vn \g_@@_right_delim_tl { . } }
2065 { \bool_gset_true:N \g_@@_delims_bool }

```

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

```
2066 \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).

```
2067 \int_compare:nNnTF \l_@@_first_col_int = 0
2068 { \tl_gput_left:NV \g_@@_preamble_tl \c_@@_preamble_first_col_tl }
2069 {
2070   \bool_lazy_all:nT
2071   {
2072     { \bool_not_p:n \g_@@_delims_bool }
2073     { \bool_not_p:n \l_@@_tabular_bool }
2074     { \tl_if_empty_p:N \l_@@_vlines_clist }
2075     { \bool_not_p:n \l_@@_exterior_arraycolsep_bool }
2076   }
2077   { \tl_gput_left:Nn \g_@@_preamble_tl { @ { } } }
2078 }
2079 \int_compare:nNnTF \l_@@_last_col_int > { -1 }
2080 { \tl_gput_right:NV \g_@@_preamble_tl \c_@@_preamble_last_col_tl }
2081 {
2082   \bool_lazy_all:nT
2083   {
2084     { \bool_not_p:n \g_@@_delims_bool }
2085     { \bool_not_p:n \l_@@_tabular_bool }
2086     { \tl_if_empty_p:N \l_@@_vlines_clist }
2087     { \bool_not_p:n \l_@@_exterior_arraycolsep_bool }
2088   }
2089   { \tl_gput_right:Nn \g_@@_preamble_tl { @ { } } }
2090 }
```

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).

```
2091 \dim_compare:nNnT \l_@@_tabular_width_dim = \c_zero_dim
2092 {
2093   \tl_gput_right:Nn \g_@@_preamble_tl
2094   { > { \@@_error_too_much_cols: } 1 }
2095 }
2096 }
```

The command \@@_patch_preamble:n is the main function for the transformation of the preamble. It is recursive.

```
2097 \cs_new_protected:Npn \@@_patch_preamble:n #1
2098 {
2099   \str_case:nnF { #1 }
2100   {
2101     c { \@@_patch_preamble_i:n #1 }
2102     l { \@@_patch_preamble_i:n #1 }
2103     r { \@@_patch_preamble_i:n #1 }
2104     > { \@@_patch_preamble_xiv:n }
2105     ! { \@@_patch_preamble_ii:nn #1 }
2106     @ { \@@_patch_preamble_ii:nn #1 }
2107     | { \@@_patch_preamble_iii:n #1 }
2108     p { \@@_patch_preamble_iv:n #1 }
2109     b { \@@_patch_preamble_iv:n #1 }
2110     m { \@@_patch_preamble_iv:n #1 }
2111     \@@_V: { \@@_patch_preamble_v:n }
2112     V { \@@_patch_preamble_v:n }
2113     \@@_w: { \@@_patch_preamble_vi:nnnn { } #1 }
2114     \@@_W: { \@@_patch_preamble_vi:nnnn { \@@_special_W: } #1 }
2115     \@@_S: { \@@_patch_preamble_vii:n }
2116     ( { \@@_patch_preamble_viii:nn #1 }
2117     [ { \@@_patch_preamble_viii:nn #1 }
```

```

2118     \{      { \@@_patch_preamble_viii:nn #1 }
2119     \left  { \@@_patch_preamble_viii:nn }
2120     )      { \@@_patch_preamble_ix:nn #1 }
2121     ]      { \@@_patch_preamble_ix:nn #1 }
2122     \}     { \@@_patch_preamble_ix:nn #1 }
2123     \right { \@@_patch_preamble_ix:nn }
2124     X      { \@@_patch_preamble_x:n }

```

When `tabularx` is loaded, a local redefinition of the specifier `X` is done to replace `X` by `\@@_X`. Thus, our column type `X` will be used in the `{NiceTabularX}`.

```

2125     \@@_X  { \@@_patch_preamble_x:n }
2126     \q_stop { }
2127   }
2128   {
2129     \str_if_eq:nVTF { #1 } \l_@@_letter_vlism_tl
2130     {
2131       \seq_gput_right:Nx \g_@@_cols_vlism_seq
2132       { \int_eval:n { \c@jCol + 1 } }
2133       \tl_gput_right:Nx \g_@@_preamble_tl
2134       { \exp_not:N ! { \skip_horizontal:N \arrayrulewidth } }
2135       \@@_patch_preamble:n
2136     }

```

Now the case of a letter set by the final user for a customized rule. Such customized rule is defined by using the key `custom-line` in `\NiceMatrixOptions`. That key takes in as value a list of *key=value* pairs. Among the keys available in that list, there is the key `letter`. All the letters defined by this way by the final user for such customized rules are added in the set of keys `{NiceMatrix/ColumnTypes}`. That set of keys is used to store the characteristics of those types of rules for convenience: the keys of that set of keys won't never be used as keys by the final user (he will use, instead, letters in the preamble of its array).

```

2137     {
2138       \keys_if_exist:nnTF { NiceMatrix / ColumnTypes } { #1 }
2139       {
2140         \keys_set:nn { NiceMatrix / ColumnTypes } { #1 }
2141         \@@_patch_preamble:n
2142       }
2143       {
2144         \tl_if_eq:nnT { #1 } { S }
2145         { \@@_fatal:n { unknown~column~type~S } }
2146         { \@@_fatal:nn { unknown~column~type } { #1 } }
2147       }
2148     }
2149   }
2150 }

```

Now, we will list all the auxiliary functions for the different types of entries in the preamble of the array.

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

```

2151 \cs_new_protected:Npn \@@_patch_preamble_i:n #1
2152 {
2153   \tl_gput_right:NV \g_@@_preamble_tl \g_@@_pre_cell_tl
2154   \tl_gclear:N \g_@@_pre_cell_tl
2155   \tl_gput_right:Nn \g_@@_preamble_tl
2156   {
2157     > { \@@_cell_begin:w \str_set:Nn \l_@@_hpos_cell_str { #1 } }
2158     #1
2159     < \@@_cell_end:
2160   }

```

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

```

2161     \int_gincr:N \c@jCol
2162     \@@_patch_preamble_xi:n
2163   }

```

For >, ! and @

```

2164 \cs_new_protected:Npn \@@_patch_preamble_ii:nn #1 #2
2165 {
2166   \tl_gput_right:Nn \g_@@_preamble_tl { #1 { #2 } }
2167   \@@_patch_preamble:n
2168 }

```

For |

```

2169 \cs_new_protected:Npn \@@_patch_preamble_iii:n #1
2170 {
\l_tmpa_int is the number of successive occurrences of |
2171   \int_incr:N \l_tmpa_int
2172   \@@_patch_preamble_iii_i:n
2173 }

2174 \cs_new_protected:Npn \@@_patch_preamble_iii_i:n #1
2175 {
2176   \str_if_eq:nnTF { #1 } |
2177   { \@@_patch_preamble_iii:n | }
2178   {
2179     \dim_set:Nn \l_tmpa_dim
2180     {
2181       \arrayrulewidth * \l_tmpa_int
2182       + \doublerulesep * ( \l_tmpa_int - 1 )
2183     }
2184     \tl_gput_right:Nx \g_@@_preamble_tl
2185     {

```

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

```

2186       \exp_not:N ! { \skip_horizontal:n { \dim_eval:n { \l_tmpa_dim } } }
2187     }
2188     \tl_gput_right:Nx \g_@@_pre_code_after_tl
2189     {
2190       \@@_vline:n
2191       {
2192         position = \int_eval:n { \c@jCol + 1 } ,
2193         multiplicity = \int_use:N \l_tmpa_int ,
2194         total-width = \dim_use:N \l_tmpa_dim
2195       }

```

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.

```

2196     }
2197     \int_zero:N \l_tmpa_int
2198     \str_if_eq:nnT { #1 } { \q_stop } { \bool_gset_true:N \g_tmpb_bool }
2199     \@@_patch_preamble:n #1
2200   }
2201 }

2202 \cs_new_protected:Npn \@@_patch_preamble_xiv:n #1
2203 {
2204   \tl_gput_right:Nn \g_@@_pre_cell_tl { > { #1 } }
2205   \@@_patch_preamble:n
2206 }

2207 \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.

```

2208 \keys_define:nn { WithArrows / p-column }
2209 {
2210   r .code:n = \str_set:Nn \l_@@_hpos_col_str { r } ,
2211   r .value_forbidden:n = true ,
2212   c .code:n = \str_set:Nn \l_@@_hpos_col_str { c } ,

```

```

2213 c .value_forbidden:n = true ,
2214 l .code:n = \str_set:Nn \l_@@_hpos_col_str { l } ,
2215 l .value_forbidden:n = true ,
2216 R .code:n =
2217   \IfPackageLoadedTF { ragged2e }
2218   { \str_set:Nn \l_@@_hpos_col_str { R } }
2219   {
2220     \@@_error_or_warning:n { ragged2e-not-loaded }
2221     \str_set:Nn \l_@@_hpos_col_str { r }
2222   } ,
2223 R .value_forbidden:n = true ,
2224 L .code:n =
2225   \IfPackageLoadedTF { ragged2e }
2226   { \str_set:Nn \l_@@_hpos_col_str { L } }
2227   {
2228     \@@_error_or_warning:n { ragged2e-not-loaded }
2229     \str_set:Nn \l_@@_hpos_col_str { l }
2230   } ,
2231 L .value_forbidden:n = true ,
2232 C .code:n =
2233   \IfPackageLoadedTF { ragged2e }
2234   { \str_set:Nn \l_@@_hpos_col_str { C } }
2235   {
2236     \@@_error_or_warning:n { ragged2e-not-loaded }
2237     \str_set:Nn \l_@@_hpos_col_str { c }
2238   } ,
2239 C .value_forbidden:n = true ,
2240 S .code:n = \str_set:Nn \l_@@_hpos_col_str { si } ,
2241 S .value_forbidden:n = true ,
2242 p .code:n = \str_set:Nn \l_@@_vpos_col_str { p } ,
2243 p .value_forbidden:n = true ,
2244 t .meta:n = p ,
2245 m .code:n = \str_set:Nn \l_@@_vpos_col_str { m } ,
2246 m .value_forbidden:n = true ,
2247 b .code:n = \str_set:Nn \l_@@_vpos_col_str { b } ,
2248 b .value_forbidden:n = true ,
2249 }

```

For `p`, `b` and `m`. The argument `#1` is that value : `p`, `b` or `m`.

```

2250 \cs_new_protected:Npn \@@_patch_preamble_iv:n #1
2251 {
2252   \str_set:Nn \l_@@_vpos_col_str { #1 }

```

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

```

2253   \@@_patch_preamble_iv_i:n
2254 }

2255 \cs_new_protected:Npn \@@_patch_preamble_iv_i:n #1
2256 {
2257   \str_if_eq:nnTF { #1 } { [ ]
2258     { \@@_patch_preamble_iv_ii:w [ ]
2259       { \@@_patch_preamble_iv_ii:w [ ] { #1 } }
2260     }

2261   \cs_new_protected:Npn \@@_patch_preamble_iv_ii:w [ #1 ]
2262     { \@@_patch_preamble_iv_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.

```

2263 \cs_new_protected:Npn \@@_patch_preamble_iv_iii:nn #1 #2
2264 {

```

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).

```

2265   \str_set:Nn \l_@@_hpos_col_str { j }

```

```

2266 \tl_set:Nn \l_tmpa_tl { #1 }
2267 \tl_replace_all:Nnn \l_tmpa_tl { \@@_S: } { S }
2268 \@@_keys_p_column:V \l_tmpa_tl
2269 \@@_patch_preamble_iv_iv:nn { #2 } { minipage }
2270 }
2271 \cs_new_protected:Npn \@@_keys_p_column:n #1
2272 { \keys_set_known:nnN { WithArrows / p-column } { #1 } \l_tmpa_tl }
2273 \cs_generate_variant:Nn \@@_keys_p_column:n { V }

```

The first argument is the width of the column. The second is the type of environment: `minipage` or `varwidth`.

```

2274 \cs_new_protected:Npn \@@_patch_preamble_iv_iv:nn #1 #2
2275 {
2276   \use:x
2277   {
2278     \@@_patch_preamble_iv_v:nnnnnnnn
2279     { \str_if_eq:VnTF \l_@@_vpos_col_str { p } { t } { b } }
2280     { \dim_eval:n { #1 } }
2281     {

```

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_str` which will provide the horizontal alignment of the column to which belongs the cell.

```

2282     \str_if_eq:VnTF \l_@@_hpos_col_str j
2283     { \str_set:Nn \exp_not:N \l_@@_hpos_cell_str { } }
2284     {
2285       \str_set:Nn \exp_not:N \l_@@_hpos_cell_str
2286       { \str_lowercase:V \l_@@_hpos_col_str }
2287     }
2288     \str_case:Vn \l_@@_hpos_col_str
2289     {
2290       c { \exp_not:N \centering }
2291       l { \exp_not:N \raggedright }
2292       r { \exp_not:N \raggedleft }
2293       C { \exp_not:N \Centering }
2294       L { \exp_not:N \RaggedRight }
2295       R { \exp_not:N \RaggedLeft }
2296     }
2297   }
2298   { \str_if_eq:VnT \l_@@_vpos_col_str { m } \@@_center_cell_box: }
2299   { \str_if_eq:VnT \l_@@_hpos_col_str { si } \siunitx_cell_begin:w }
2300   { \str_if_eq:VnT \l_@@_hpos_col_str { si } \siunitx_cell_end: }
2301   { #2 }
2302   {
2303     \str_case:VnF \l_@@_hpos_col_str
2304     {
2305       { j } { c }
2306       { si } { c }
2307     }

```

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

```

2308     { \str_lowercase:V \l_@@_hpos_col_str }
2309   }
2310 }

```

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

```

2311 \int_gincr:N \c@jCol
2312 \@@_patch_preamble_xi:n
2313 }

```

`#1` is the optional argument of `{minipage}` (or `{varwidth}`): `t` of `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_str 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*.

```

2314 \cs_new_protected:Npn \@@_patch_preamble_iv_v:nnnnnnnn #1 #2 #3 #4 #5 #6 #7 #8
2315 {
2316   \str_if_eq:VnTF \l_@@_hpos_col_str { si }
2317     { \tl_gput_right:Nn \g_@@_preamble_tl { > { \@@_test_if_empty_for_S: } } }
2318     { \tl_gput_right:Nn \g_@@_preamble_tl { > { \@@_test_if_empty: } } }
2319   \tl_gput_right:NV \g_@@_preamble_tl \g_@@_pre_cell_tl
2320   \tl_gclear:N \g_@@_pre_cell_tl
2321   \tl_gput_right:Nn \g_@@_preamble_tl
2322     {
2323     > {

```

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.

```

2324     \dim_set:Nn \l_@@_col_width_dim { #2 }
2325     \@@_cell_begin:w
2326     \begin { #7 } [ #1 ] { #2 }

```

The following lines have been taken from array.sty.

```

2327     \everypar
2328     {
2329         \vrule height \box_ht:N \@arstrutbox width \c_zero_dim
2330         \everypar { }
2331     }

```

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

```

2332     #3

```

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

```

2333     \g_@@_row_style_tl
2334     \arraybackslash
2335     #5
2336   }
2337   #8
2338   < {
2339     #6

```

The following line has been taken from array.sty.

```

2340     \@finalstrut \@arstrutbox
2341     % \bool_if:NT \g_@@_rotate_bool { \raggedright \hspace = 3 cm }
2342     \end { #7 }

```

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

```

2343     #4
2344     \@@_cell_end:
2345   }
2346 }
2347 }

```

```

2348 \cs_new_protected:Npn \@@_test_if_empty: \ignorespaces #1
2349 {
2350   \peek_meaning:NT \unskip
2351   {
2352     \tl_gput_right:Nn \g_@@_cell_after_hook_tl

```

```

2353     {
2354     \box_set_wd:Nn \l_@@_cell_box \c_zero_dim

```

We put the following code in order to have a column with the correct width even when all the cells of the column are empty.

```

2355     \skip_horizontal:N \l_@@_col_width_dim
2356     }
2357   }
2358   #1
2359 }

2360 \cs_new_protected:Npn \@@_test_if_empty_for_S: #1
2361 {
2362   \peek_meaning:NT \_siunitx_table_skip:n
2363   {
2364     \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2365     { \box_set_wd:Nn \l_@@_cell_box \c_zero_dim }
2366   }
2367   #1
2368 }

```

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.

```

2369 \cs_new_protected:Npn \@@_center_cell_box:
2370 {

```

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

```

2371   \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2372   {
2373     \int_compare:nNnT
2374     { \box_ht:N \l_@@_cell_box }
2375     >

```

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).

```

2376     { \box_ht:N \strutbox }
2377     {
2378       \hbox_set:Nn \l_@@_cell_box
2379       {
2380         \box_move_down:nn
2381         {
2382           ( \box_ht:N \l_@@_cell_box - \box_ht:N \@arstrutbox
2383             + \baselineskip ) / 2
2384         }
2385         { \box_use:N \l_@@_cell_box }
2386       }
2387     }
2388   }
2389 }

```

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

```

2390 \cs_new_protected:Npn \@@_patch_preamble_v:n #1
2391 {
2392   \str_if_eq:nnTF { #1 } { [ ]
2393     { \@@_patch_preamble_v_i:w [ ]
2394       { \@@_patch_preamble_v_i:w [ ] { #1 } }
2395     }
2396   \cs_new_protected:Npn \@@_patch_preamble_v_i:w [ #1 ]

```



```

2397 { \@@_patch_preamble_v_ii:nn { #1 } }
2398 \cs_new_protected:Npn \@@_patch_preamble_v_ii:nn #1 #2
2399 {
2400   \str_set:Nn \l_@@_vpos_col_str { p }
2401   \str_set:Nn \l_@@_hpos_col_str { j }
2402   \tl_set:Nn \l_tmpa_tl { #1 }
2403   \tl_replace_all:Nnn \l_tmpa_tl { \@@_S: } { S }
2404   \@@_keys_p_column:V \l_tmpa_tl
2405   \IfPackageLoadedTF { varwidth }
2406     { \@@_patch_preamble_iv_iv:nn { #2 } { varwidth } }
2407     {
2408       \@@_error_or_warning:n { varwidth~not~loaded }
2409       \@@_patch_preamble_iv_iv:nn { #2 } { minipage }
2410     }
2411 }

```

For w and 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.

```

2412 \cs_new_protected:Npn \@@_patch_preamble_vi:nnnn #1 #2 #3 #4
2413 {
2414   \str_if_eq:nnTF { #3 } { s }
2415     { \@@_patch_preamble_vi_i:nnnn { #1 } { #4 } }
2416     { \@@_patch_preamble_vi_ii:nnnn { #1 } { #2 } { #3 } { #4 } }
2417 }

```

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.

```

2418 \cs_new_protected:Npn \@@_patch_preamble_vi_i:nnnn #1 #2
2419 {
2420   \tl_gput_right:NV \g_@@_preamble_tl \g_@@_pre_cell_tl
2421   \tl_gclear:N \g_@@_pre_cell_tl
2422   \tl_gput_right:Nn \g_@@_preamble_tl
2423     {
2424       > {
2425         \dim_set:Nn \l_@@_col_width_dim { #2 }
2426         \@@_cell_begin:w
2427         \str_set:Nn \l_@@_hpos_cell_str { c }
2428       }
2429       c
2430       < {
2431         \@@_cell_end_for_w_s:
2432         #1
2433         \@@_adjust_size_box:
2434         \box_use_drop:N \l_@@_cell_box
2435       }
2436     }
2437   \int_gincr:N \c@jCol
2438   \@@_patch_preamble_xi:n
2439 }

```

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

```

2440 \cs_new_protected:Npn \@@_patch_preamble_vi_ii:nnnn #1 #2 #3 #4
2441 {
2442   \tl_gput_right:NV \g_@@_preamble_tl \g_@@_pre_cell_tl
2443   \tl_gclear:N \g_@@_pre_cell_tl
2444   \tl_gput_right:Nn \g_@@_preamble_tl
2445     {
2446       > {

```

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.

```

2447         \dim_set:Nn \l_@@_col_width_dim { #4 }
2448         \hbox_set:Nw \l_@@_cell_box
2449         \@@_cell_begin:w
2450         \str_set:Nn \l_@@_hpos_cell_str { #3 }
2451     }
2452     c
2453     < {
2454         \@@_cell_end:
2455         \hbox_set_end:
2456         #1
2457         \@@_adjust_size_box:
2458         \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2459     }
2460 }

```

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

```

2461     \int_gincr:N \c@jCol
2462     \@@_patch_preamble_xi:n
2463 }

```

```

2464 \cs_new_protected:Npn \@@_special_W:
2465 {
2466     \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \l_@@_col_width_dim
2467     { \@@_warning:n { W~warning } }
2468 }

```

For `\@@_S:`. If the user has used `S[...]`, `S` has been replaced by `\@@_S:` during the first expansion of the preamble (done with the tools of standard LaTeX and array).

```

2469 \cs_new_protected:Npn \@@_patch_preamble_vii:n #1
2470 {
2471     \str_if_eq:nnTF { #1 } { [ ]
2472     { \@@_patch_preamble_vii_i:w [ ]
2473     { \@@_patch_preamble_vii_i:w [ ] { #1 } }
2474 }
2475 \cs_new_protected:Npn \@@_patch_preamble_vii_i:w [ #1 ]
2476 { \@@_patch_preamble_vii_ii:n { #1 } }
2477 \cs_new_protected:Npn \@@_patch_preamble_vii_ii:n #1
2478 {
2479     \IfPackageAtLeastTF { siunitx } { 2022/01/01 }
2480     {
2481         \tl_gput_right:NV \g_@@_preamble_tl \g_@@_pre_cell_tl
2482         \tl_gclear:N \g_@@_pre_cell_tl
2483         \tl_gput_right:Nn \g_@@_preamble_tl
2484         {
2485             > {
2486                 \@@_cell_begin:w
2487                 \keys_set:nn { siunitx } { #1 }
2488                 \siunitx_cell_begin:w
2489             }
2490             c
2491             < { \siunitx_cell_end: \@@_cell_end: }
2492         }

```

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

```

2493     \int_gincr:N \c@jCol
2494     \@@_patch_preamble_xi:n
2495 }
2496 { \@@_fatal:n { Version~of~siunitx~too~old } }
2497 }

```

For (, [and \{.

```

2498 \cs_new_protected:Npn \@@_patch_preamble_viii:nn #1 #2
2499   {
2500     \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.

```

2501     \int_compare:nNnTF \c@jCol = \c_zero_int
2502     {
2503       \str_if_eq:VnTF \g_@@_left_delim_tl { . }
2504       {

```

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

```

2505         \tl_gset:Nn \g_@@_left_delim_tl { #1 }
2506         \tl_gset:Nn \g_@@_right_delim_tl { . }
2507         \@@_patch_preamble:n #2
2508       }
2509     {
2510       \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } }
2511       \@@_patch_preamble_viii_i:nn { #1 } { #2 }
2512     }
2513   }
2514   { \@@_patch_preamble_viii_i:nn { #1 } { #2 } }
2515 }

```

```

2516 \cs_new_protected:Npn \@@_patch_preamble_viii_i:nn #1 #2
2517   {
2518     \tl_gput_right:Nx \g_@@_pre_code_after_tl
2519     { \@@_delimiter:nnn #1 { \int_eval:n { \c@jCol + 1 } } \c_true_bool }
2520     \tl_if_in:nnTF { ( [ \{ ) ] \} \left \right } { #2 }
2521     {
2522       \@@_error:nn { delimiter-after-opening } { #2 }
2523       \@@_patch_preamble:n
2524     }
2525     { \@@_patch_preamble:n #2 }
2526   }

```

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

```

2527 \cs_new_protected:Npn \@@_patch_preamble_ix:nn #1 #2
2528   {
2529     \bool_if:NT \l_@@_small_bool { \@@_fatal:n { Delimiter~with~small } }
2530     \tl_if_in:nnTF { ) ] \} } { #2 }
2531     { \@@_patch_preamble_ix_i:nnn #1 #2 }
2532     {
2533       \tl_if_eq:nnTF { \q_stop } { #2 }
2534       {
2535         \str_if_eq:VnTF \g_@@_right_delim_tl { . }
2536         { \tl_gset:Nn \g_@@_right_delim_tl { #1 } }
2537         {
2538           \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } }
2539           \tl_gput_right:Nx \g_@@_pre_code_after_tl
2540           { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2541           \@@_patch_preamble:n #2
2542         }
2543       }
2544     }
2545     \tl_if_in:nnT { ( [ \{ \left } { #2 }
2546     { \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } } }
2547     \tl_gput_right:Nx \g_@@_pre_code_after_tl
2548     { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2549     \@@_patch_preamble:n #2

```

```

2550     }
2551   }
2552 }
2553 \cs_new_protected:Npn \@@_patch_preamble_ix_i:nnn #1 #2 #3
2554 {
2555   \tl_if_eq:nnTF { \q_stop } { #3 }
2556   {
2557     \str_if_eq:VnTF \g_@@_right_delim_tl { . }
2558     {
2559       \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } }
2560       \tl_gput_right:Nx \g_@@_pre_code_after_tl
2561         { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2562       \tl_gset:Nn \g_@@_right_delim_tl { #2 }
2563     }
2564     {
2565       \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } }
2566       \tl_gput_right:Nx \g_@@_pre_code_after_tl
2567         { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2568       \@@_error:nn { double-closing-delimiter } { #2 }
2569     }
2570   }
2571   {
2572     \tl_gput_right:Nx \g_@@_pre_code_after_tl
2573       { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2574     \@@_error:nn { double-closing-delimiter } { #2 }
2575     \@@_patch_preamble:n #3
2576   }
2577 }

```

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.

```

2578 \cs_new_protected:Npn \@@_patch_preamble_x:n #1
2579 {
2580   \str_if_eq:nnTF { #1 } { [ ]
2581     { \@@_patch_preamble_x_i:w [ ]
2582       { \@@_patch_preamble_x_i:w [ ] #1 }
2583     }
2584   \cs_new_protected:Npn \@@_patch_preamble_x_i:w [ #1 ]
2585     { \@@_patch_preamble_x_ii: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`).

```

2586 \keys_define:nn { WithArrows / X-column }
2587   { 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.

```

2588 \cs_new_protected:Npn \@@_patch_preamble_x_ii:n #1
2589 {

```

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).

```

2590   \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).

```

2591   \tl_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`.

```

2592 \int_zero_new:N \l_@@_weight_int
2593 \int_set:Nn \l_@@_weight_int { 1 }
2594 \tl_set:Nn \l_tmpa_tl { #1 }
2595 \tl_replace_all:Nnn \l_tmpa_tl { \@@_S: } { S }
2596 \@@_keys_p_column:V \l_tmpa_tl
2597 \keys_set:nV { WithArrows / X-column } \l_tmpa_tl
2598 \int_compare:nNnT \l_@@_weight_int < 0
2599 {
2600   \@@_error_or_warning:n { negative~weight }
2601   \int_set:Nn \l_@@_weight_int { - \l_@@_weight_int }
2602 }
2603 \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).

```

2604 \bool_if:NTF \l_@@_X_columns_aux_bool
2605 {
2606   \exp_args:Nnx
2607   \@@_patch_preamble_iv_iv:nn
2608   { \l_@@_weight_int \l_@@_X_columns_dim }
2609   { minipage }
2610 }
2611 {
2612   \tl_gput_right:Nn \g_@@_preamble_tl
2613   {
2614     > {
2615       \@@_cell_begin:w
2616       \bool_set_true:N \l_@@_X_column_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`.

```

2617 \NotEmpty

```

The following code will nullify the box of the cell.

```

2618 \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2619 { \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`.

```

2620 \begin { minipage } { 5 cm } \arraybackslash
2621 }
2622 c
2623 < {
2624   \end { minipage }
2625   \@@_cell_end:
2626 }
2627 }
2628 \int_gincr:N \c@jCol
2629 \@@_patch_preamble_xi:n
2630 }
2631 }

```

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 `@{...}`.

```

2632 \cs_new_protected:Npn \@@_patch_preamble_xi:n #1
2633 {

```

```

2634 \str_if_eq:nnTF { #1 } { < }
2635 \@@_patch_preamble_xiii:n
2636 {
2637   \str_if_eq:nnTF { #1 } { @ }
2638   \@@_patch_preamble_xv:n
2639   {
2640     \tl_if_eq:NnTF \l_@@_vlines_clist { all }
2641     {
2642       \tl_gput_right:Nn \g_@@_preamble_tl
2643       { ! { \skip_horizontal:N \arrayrulewidth } }
2644     }
2645     {
2646       \exp_args:NNx
2647       \clist_if_in:NnT \l_@@_vlines_clist { \int_eval:n { \c@jCol + 1 } }
2648       {
2649         \tl_gput_right:Nn \g_@@_preamble_tl
2650         { ! { \skip_horizontal:N \arrayrulewidth } }
2651       }
2652     }
2653     \@@_patch_preamble:n { #1 }
2654   }
2655 }
2656 }
2657 \cs_new_protected:Npn \@@_patch_preamble_xiii:n #1
2658 {
2659   \tl_gput_right:Nn \g_@@_preamble_tl { < { #1 } }
2660   \@@_patch_preamble_xi:n
2661 }

```

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.

```

2662 \cs_new_protected:Npn \@@_patch_preamble_xv:n #1
2663 {
2664   \tl_if_eq:NnTF \l_@@_vlines_clist { all }
2665   {
2666     \tl_gput_right:Nn \g_@@_preamble_tl
2667     { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2668   }
2669   {
2670     \exp_args:NNx
2671     \clist_if_in:NnTF \l_@@_vlines_clist { \int_eval:n { \c@jCol + 1 } }
2672     {
2673       \tl_gput_right:Nn \g_@@_preamble_tl
2674       { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2675     }
2676     { \tl_gput_right:Nn \g_@@_preamble_tl { @ { #1 } } }
2677   }
2678   \@@_patch_preamble:n
2679 }
2680 \cs_new_protected:Npn \@@_set_preamble:Nn #1 #2
2681 {
2682   \group_begin:
2683   \@@_newcolumnntype w [ 2 ] { \@@_w: { ##1 } { ##2 } }
2684   \@@_newcolumnntype W [ 2 ] { \@@_W: { ##1 } { ##2 } }
2685   \@temptokena { #2 }
2686   \@tempswatruw
2687   \@whilesw \if@tempswa \fi { \@tempswafalse \the \NC@list }
2688   \tl_gclear:N \g_@@_preamble_tl
2689   \exp_after:wN \@@_patch_m_preamble:n \the \@temptokena \q_stop
2690   \group_end:
2691   \tl_set_eq:NN #1 \g_@@_preamble_tl
2692 }

```

13 The redefinition of `\multicolumn`

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

```
2693 \cs_new:Npn \@_multicolumn:nnn #1 #2 #3
2694 {
```

The following lines are from the definition of `\multicolumn` in `array` (and *not* in standard LaTeX). The first line aims to raise an error if the user has put more than one column specifier in the preamble of `\multicolumn`.

```
2695 \multispan { #1 }
2696 \begingroup
2697 \cs_set:Npn \@addamp { \if@firstamp \@firstampfalse \else \@preammerr 5 \fi }
2698 \@_newcolumnntype w [ 2 ] { \@_w: { ##1 } { ##2 } }
2699 \@_newcolumnntype W [ 2 ] { \@_W: { ##1 } { ##2 } }
```

You do the expansion of the (small) preamble with the tools of `array`.

```
2700 \@temptokena = { #2 }
2701 \@tempswatrue
2702 \@whilesw \if@tempswa \fi { \@tempswafalse \the \NC@list }
```

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

```
2703 \tl_gclear:N \g_@@_preamble_tl
2704 \exp_after:wN \@_patch_m_preamble:n \the \@temptokena \q_stop
```

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

```
2705 \exp_args:NV \@mkpream \g_@@_preamble_tl
2706 \@addtopreamble \@empty
2707 \endgroup
```

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

```
2708 \int_compare:nNnT { #1 } > 1
2709 {
2710   \seq_gput_left:Nx \g_@@_multicolumn_cells_seq
2711     { \int_use:N \c@iRow - \int_eval:n { \c@jCol + 1 } }
2712   \seq_gput_left:Nn \g_@@_multicolumn_sizes_seq { #1 }
2713   \seq_gput_right:Nx \g_@@_pos_of_blocks_seq
2714     {
2715       {
2716         \int_compare:nNnTF \c@jCol = 0
2717           { \int_eval:n { \c@iRow + 1 } }
2718           { \int_use:N \c@iRow }
2719       }
2720     { \int_eval:n { \c@jCol + 1 } }
2721     {
2722       \int_compare:nNnTF \c@jCol = 0
2723         { \int_eval:n { \c@iRow + 1 } }
2724         { \int_use:N \c@iRow }
2725     }
2726     { \int_eval:n { \c@jCol + #1 } }
2727     { } % for the name of the block
2728   }
2729 }
```

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

```
2730 \cs_set:Npn \@sharp { #3 }
2731 \@arstrut
2732 \@preamble
2733 \null
```

We add some lines.

```

2734 \int_gadd:Nn \c@jCol { #1 - 1 }
2735 \int_compare:nNnT \c@jCol > \g_@@_col_total_int
2736   { \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
2737 \ignorespaces
2738 }

```

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`.

```

2739 \cs_new_protected:Npn \@@_patch_m_preamble:n #1
2740 {
2741   \str_case:nnF { #1 }
2742   {
2743     c { \@@_patch_m_preamble_i:n #1 }
2744     l { \@@_patch_m_preamble_i:n #1 }
2745     r { \@@_patch_m_preamble_i:n #1 }
2746     > { \@@_patch_m_preamble_ii:nn #1 }
2747     ! { \@@_patch_m_preamble_ii:nn #1 }
2748     @ { \@@_patch_m_preamble_ii:nn #1 }
2749     | { \@@_patch_m_preamble_iii:n #1 }
2750     p { \@@_patch_m_preamble_iv:nnn t #1 }
2751     m { \@@_patch_m_preamble_iv:nnn c #1 }
2752     b { \@@_patch_m_preamble_iv:nnn b #1 }
2753     \@@_w: { \@@_patch_m_preamble_v:nnnn { } #1 }
2754     \@@_W: { \@@_patch_m_preamble_v:nnnn { \@@_special_W: } #1 }
2755     \q_stop { }
2756   }
2757   {
2758     \tl_if_eq:nnT { #1 } { S }
2759     { \@@_fatal:n { unknown-column-type~S } }
2760     { \@@_fatal:nn { unknown-column-type } { #1 } }
2761   }
2762 }

```

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

```

2763 \cs_new_protected:Npn \@@_patch_m_preamble_i:n #1
2764 {
2765   \tl_gput_right:Nn \g_@@_preamble_tl
2766   {
2767     > { \@@_cell_begin:w \str_set:Nn \l_@@_hpos_cell_str { #1 } }
2768     #1
2769     < \@@_cell_end:
2770   }

```

We test for the presence of a `<`.

```

2771 \@@_patch_m_preamble_x:n
2772 }

```

For `>`, `!` and `@`

```

2773 \cs_new_protected:Npn \@@_patch_m_preamble_ii:nn #1 #2
2774 {
2775   \tl_gput_right:Nn \g_@@_preamble_tl { #1 { #2 } }
2776   \@@_patch_m_preamble:n
2777 }

```

For `|`

```

2778 \cs_new_protected:Npn \@@_patch_m_preamble_iii:n #1
2779 {
2780   \tl_gput_right:Nn \g_@@_preamble_tl { #1 }
2781   \@@_patch_m_preamble:n
2782 }

```


For p, m and b

```
2783 \cs_new_protected:Npn \@@_patch_m_preamble_iv:nnn #1 #2 #3
2784 {
2785   \tl_gput_right:Nn \g_@@_preamble_tl
2786   {
2787     > {
2788       \@@_cell_begin:w
2789       \begin { minipage } [ #1 ] { \dim_eval:n { #3 } }
2790       \mode_leave_vertical:
2791       \arraybackslash
2792       \vrule height \box_ht:N \@arstrutbox depth 0 pt width 0 pt
2793     }
2794     c
2795     < {
2796       \vrule height 0 pt depth \box_dp:N \@arstrutbox width 0 pt
2797       \end { minipage }
2798       \@@_cell_end:
2799     }
2800   }
```

We test for the presence of a <.

```
2801   \@@_patch_m_preamble_x:n
2802 }
```

For w and W

```
2803 \cs_new_protected:Npn \@@_patch_m_preamble_v:nmmm #1 #2 #3 #4
2804 {
2805   \tl_gput_right:Nn \g_@@_preamble_tl
2806   {
2807     > {
2808       \dim_set:Nn \l_@@_col_width_dim { #4 }
2809       \hbox_set:Nw \l_@@_cell_box
2810       \@@_cell_begin:w
2811       \str_set:Nn \l_@@_hpos_cell_str { #3 }
2812     }
2813     c
2814     < {
2815       \@@_cell_end:
2816       \hbox_set_end:
2817       \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
2818       #1
2819       \@@_adjust_size_box:
2820       \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2821     }
2822   }
```

We test for the presence of a <.

```
2823   \@@_patch_m_preamble_x:n
2824 }
```

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

```
2825 \cs_new_protected:Npn \@@_patch_m_preamble_x:n #1
2826 {
2827   \str_if_eq:nnTF { #1 } { < }
2828   \@@_patch_m_preamble_ix:n
2829   { \@@_patch_m_preamble:n { #1 } }
2830 }
2831 \cs_new_protected:Npn \@@_patch_m_preamble_ix:n #1
2832 {
2833   \tl_gput_right:Nn \g_@@_preamble_tl { < { #1 } }
2834   \@@_patch_m_preamble_x:n
2835 }
```

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`).

```

2836 \cs_new_protected:Npn \@@_put_box_in_flow:
2837   {
2838     \box_set_ht:Nn \l_tmpa_box { \box_ht:N \l_tmpa_box + \l_tmpa_dim }
2839     \box_set_dp:Nn \l_tmpa_box { \box_dp:N \l_tmpa_box + \l_tmpb_dim }
2840     \tl_if_eq:NnTF \l_@@_baseline_tl { c }
2841       { \box_use_drop:N \l_tmpa_box }
2842       \@@_put_box_in_flow_i:
2843   }

```

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).

```

2844 \cs_new_protected:Npn \@@_put_box_in_flow_i:
2845   {
2846     \pgfpicture
2847       \@@_qpoint:n { row - 1 }
2848       \dim_gset_eq:NN \g_tmpa_dim \pgf@y
2849       \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
2850       \dim_gadd:Nn \g_tmpa_dim \pgf@y
2851       \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).

```

2852     \str_if_in:NnTF \l_@@_baseline_tl { line- }
2853       {
2854         \int_set:Nn \l_tmpa_int
2855           {
2856             \str_range:Nnn
2857               \l_@@_baseline_tl
2858               6
2859               { \tl_count:V \l_@@_baseline_tl }
2860           }
2861         \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
2862       }
2863     {
2864       \str_case:VnF \l_@@_baseline_tl
2865         {
2866           { t } { \int_set:Nn \l_tmpa_int 1 }
2867           { b } { \int_set_eq:NN \l_tmpa_int \c@iRow }
2868         }
2869       { \int_set:Nn \l_tmpa_int \l_@@_baseline_tl }
2870     \bool_lazy_or:nnT
2871       { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
2872       { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
2873     {
2874       \@@_error:n { bad~value~for~baseline }
2875       \int_set:Nn \l_tmpa_int 1
2876     }
2877     \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }

```

We take into account the position of the mathematical axis.

```

2878     \dim_gsub:Nn \g_tmpa_dim { \fontdimen22 \textfont2 }
2879   }
2880   \dim_gsub:Nn \g_tmpa_dim \pgf@y

```

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

```

2881   \endpgfpicture
2882   \box_move_up:nn \g_tmpa_dim { \box_use_drop:N \l_tmpa_box }
2883   \box_use_drop:N \l_tmpa_box
2884 }

```

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).

```
2885 \cs_new_protected:Npn \@@_use_arraybox_with_notes_c:
2886 {
```

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.

```
2887   \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
2888     {
2889       \int_compare:nNnT \c@jCol > 1 % added 2023-08-13
2890         {
2891           \box_set_wd:Nn \l_@@_the_array_box
2892             { \box_wd:N \l_@@_the_array_box - \arraycolsep }
2893         }
2894     }
```

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

```
2895   \begin { minipage } [ t ] { \box_wd:N \l_@@_the_array_box }
2896   \bool_if:NT \l_@@_caption_above_bool
2897     {
2898       \tl_if_empty:NF \l_@@_caption_tl
2899         {
2900           \bool_set_false:N \g_@@_caption_finished_bool
2901           \int_gzero:N \c@tabularnote
2902           \@@_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).

```
2903           \int_compare:nNnT \g_@@_notes_caption_int > 0
2904             {
2905               \tl_gput_right:Nx \g_@@_aux_tl
2906                 {
2907                   \tl_set:Nn \exp_not:N \l_@@_note_in_caption_tl
2908                     { \int_use:N \g_@@_notes_caption_int }
2909                 }
2910               \int_gzero:N \g_@@_notes_caption_int
2911             }
2912         }
2913     }
```

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

```
2914   \hbox
2915     {
2916       \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.

```
2917       \@@_create_extra_nodes:
2918       \seq_if_empty:NF \g_@@_blocks_seq \@@_draw_blocks:
2919     }
```

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

```
2920   \bool_lazy_any:nT
2921     {
2922       { ! \seq_if_empty_p:N \g_@@_notes_seq }
```

```

2923     { ! \seq_if_empty_p:N \g_@@_notes_in_caption_seq }
2924     { ! \tl_if_empty_p:V \g_@@_tabularnote_tl }
2925   }
2926   \@@_insert_tabularnotes:
2927   \cs_set_eq:NN \tabularnote \@@_tabularnote_error:n
2928   \bool_if:NF \l_@@_caption_above_bool \@@_insert_caption:
2929   \end { minipage }
2930 }

```

```

2931 \cs_new_protected:Npn \@@_insert_caption:
2932 {
2933   \tl_if_empty:NF \l_@@_caption_tl
2934   {
2935     \cs_if_exist:NTF \@capttype
2936     { \@@_insert_caption_i: }
2937     { \@@_error:n { caption-outside~float } }
2938   }
2939 }

```

```

2940 \cs_new_protected:Npn \@@_insert_caption_i:
2941 {
2942   \group_begin:

```

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

```

2943   \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.

```

2944   \IfPackageLoadedTF { floatrow }
2945   { \cs_set_eq:NN \@makecaption \FR@makecaption }
2946   { }
2947   \tl_if_empty:NTF \l_@@_short_caption_tl
2948   { \caption }
2949   { \caption [ \l_@@_short_caption_tl ] }
2950   { \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.

```

2951   \bool_if:NF \g_@@_caption_finished_bool % added 2023/06/30
2952   {
2953     \bool_gset_true:N \g_@@_caption_finished_bool
2954     \int_gset_eq:NN \g_@@_notes_caption_int \c@tabularnote
2955     \int_gzero:N \c@tabularnote
2956   }
2957   \tl_if_empty:NF \l_@@_label_tl { \label { \l_@@_label_tl } }
2958   \group_end:
2959 }

```

```

2960 \cs_new_protected:Npn \@@_tabularnote_error:n #1
2961 {
2962   \@@_error_or_warning:n { tabularnote~below~the~tabular }
2963   \@@_gredirect_none:n { tabularnote~below~the~tabular }
2964 }

```

```

2965 \cs_new_protected:Npn \@@_insert_tabularnotes:
2966 {
2967   \seq_gconcat:NNN \g_@@_notes_seq \g_@@_notes_in_caption_seq \g_@@_notes_seq
2968   \int_set:Nn \c@tabularnote { \seq_count:N \g_@@_notes_seq }
2969   \skip_vertical:N 0.65ex

```

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

```

2970 \group_begin:
2971 \l_@@_notes_code_before_tl
2972 \tl_if_empty:NF \g_@@_tabularnote_tl
2973 {
2974   \g_@@_tabularnote_tl \par
2975   \tl_gclear:N \g_@@_tabularnote_tl
2976 }

```

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.

```

2977 \int_compare:nNnT \c@tabularnote > 0
2978 {
2979   \bool_if:NTF \l_@@_notes_para_bool
2980   {
2981     \begin { tabularnotes* }
2982     \seq_map_inline:Nn \g_@@_notes_seq
2983       { \@@_one_tabularnote:nn ##1 }
2984     \strut
2985     \end { tabularnotes* }

```

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

```

2986   \par
2987 }
2988 {
2989   \tabularnotes
2990   \seq_map_inline:Nn \g_@@_notes_seq
2991     { \@@_one_tabularnote:nn ##1 }
2992   \strut
2993   \endtabularnotes
2994 }
2995 }
2996 \unskip
2997 \group_end:
2998 \bool_if:NT \l_@@_notes_bottomrule_bool
2999 {
3000   \IfPackageLoadedTF { booktabs }
3001   {

```

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

```

3002   \skip_vertical:N \aboverulesep

```

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

```

3003   { \CT@arc@ \hrule height \heavyrulewidth }
3004 }
3005 { \@@_error_or_warning:n { bottomrule-without-booktabs } }
3006 }
3007 \l_@@_notes_code_after_tl
3008 \seq_gclear:N \g_@@_notes_seq
3009 \seq_gclear:N \g_@@_notes_in_caption_seq
3010 \int_gzero:N \c@tabularnote
3011 }

```

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.

```

3012 \cs_set_protected:Npn \@@_one_tabularnote:nn #1
3013 {
3014   \tl_if_novalue:nTF { #1 }
3015     { \item }
3016     { \item [ \@@_notes_label_in_list:n { #1 } ] }
3017 }

```

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`.

```

3018 \cs_new_protected:Npn \@@_use_arraybox_with_notes_b:
3019 {
3020   \pgfpicture
3021     \@@_qpoint:n { row - 1 }
3022     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3023     \@@_qpoint:n { row - \int_use:N \c@iRow - base }
3024     \dim_gsub:Nn \g_tmpa_dim \pgf@y
3025   \endpgfpicture
3026   \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3027   \int_compare:nNnT \l_@@_first_row_int = 0
3028     {
3029       \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3030       \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3031     }
3032   \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3033 }

```

Now, the general case.

```

3034 \cs_new_protected:Npn \@@_use_arraybox_with_notes:
3035 {

```

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

```

3036   \tl_if_eq:NnT \l_@@_baseline_tl { t }
3037     { \tl_set:Nn \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`.

```

3038   \pgfpicture
3039   \@@_qpoint:n { row - 1 }
3040   \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3041   \str_if_in:NnTF \l_@@_baseline_tl { line- }
3042     {
3043       \int_set:Nn \l_tmpa_int
3044         {
3045           \str_range:Nnn
3046             \l_@@_baseline_tl
3047             6
3048             { \tl_count:V \l_@@_baseline_tl }
3049         }
3050       \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
3051     }
3052   {
3053     \int_set:Nn \l_tmpa_int \l_@@_baseline_tl
3054     \bool_lazy_or:nnT
3055       { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
3056       { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
3057     {
3058       \@@_error:n { bad-value-for-baseline }
3059       \int_set:Nn \l_tmpa_int 1
3060     }
3061     \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }
3062   }
3063   \dim_gsub:Nn \g_tmpa_dim \pgf@y
3064   \endpgfpicture
3065   \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3066   \int_compare:nNnT \l_@@_first_row_int = 0
3067     {
3068       \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3069       \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3070     }
3071   \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }

```

```
3072 }
```

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.

```
3073 \cs_new_protected:Npn \@@_put_box_in_flow_bis:nn #1 #2
3074 {
```

We will compute the real width of both delimiters used.

```
3075 \dim_zero_new:N \l_@@_real_left_delim_dim
3076 \dim_zero_new:N \l_@@_real_right_delim_dim
3077 \hbox_set:Nn \l_tmpb_box
3078 {
3079   \c_math_toggle_token
3080   \left #1
3081   \vcenter
3082   {
3083     \vbox_to_ht:nn
3084     { \box_ht_plus_dp:N \l_tmpa_box }
3085     { }
3086   }
3087   \right .
3088   \c_math_toggle_token
3089 }
3090 \dim_set:Nn \l_@@_real_left_delim_dim
3091 { \box_wd:N \l_tmpb_box - \nulldelimiterspace }
3092 \hbox_set:Nn \l_tmpb_box
3093 {
3094   \c_math_toggle_token
3095   \left .
3096   \vbox_to_ht:nn
3097   { \box_ht_plus_dp:N \l_tmpa_box }
3098   { }
3099   \right #2
3100   \c_math_toggle_token
3101 }
3102 \dim_set:Nn \l_@@_real_right_delim_dim
3103 { \box_wd:N \l_tmpb_box - \nulldelimiterspace }
```

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

```
3104 \skip_horizontal:N \l_@@_left_delim_dim
3105 \skip_horizontal:N -\l_@@_real_left_delim_dim
3106 \@@_put_box_in_flow:
3107 \skip_horizontal:N \l_@@_right_delim_dim
3108 \skip_horizontal:N -\l_@@_real_right_delim_dim
3109 }
```

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).

```
3110 \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).

```
3111 {
3112   \peek_remove_spaces:n
3113   {
3114     \peek_meaning:NTF \end
3115     \@@_analyze_end:Nn
```

```

3116     {
3117     \@@_transform_preamble:
Here is the call to \array (we have a dedicated macro \@@_array:n because of compatibility with
the classes revtex4-1 and revtex4-2).
3118     \@@_array:V \g_@@_preamble_tl
3119     }
3120   }
3121 }
3122 {
3123 \@@_create_col_nodes:
3124 \endarray
3125 }

```

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

```

3126 \NewDocumentEnvironment { @@-light-syntax } { b }
3127 {

```

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`.

```

3128 \tl_if_empty:nT { #1 } { \@@_fatal:n { empty-environment } }
3129 \tl_map_inline:nn { #1 }
3130 {
3131   \str_if_eq:nnT { ##1 } { & }
3132   { \@@_fatal:n { ampersand-in-light-syntax } }
3133   \str_if_eq:nnT { ##1 } { \ }
3134   { \@@_fatal:n { double-backslash-in-light-syntax } }
3135 }

```

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`.

```

3136 \@@_light_syntax_i:w #1 \CodeAfter \q_stop

```

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

```

3137 }

```

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.

```

3138 {
3139 \@@_create_col_nodes:
3140 \endarray
3141 }
3142 \cs_new_protected:Npn \@@_light_syntax_i:w #1\CodeAfter #2\q_stop
3143 {
3144 \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).

```

3145 \seq_clear_new:N \l_@@_rows_seq

```

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

```

3146 \tl_set_rescan:Nno \l_@@_end_of_row_tl { } \l_@@_end_of_row_tl
3147 \seq_set_split:NvN \l_@@_rows_seq \l_@@_end_of_row_tl { #1 }

```

We delete the last row if it is empty.

```

3148 \seq_pop_right:NN \l_@@_rows_seq \l_tmpa_tl
3149 \tl_if_empty:NF \l_tmpa_tl
3150 { \seq_put_right:Nv \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.

```
3151 \int_compare:nNnT \l_@@_last_row_int = { -1 }
3152 { \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` (that part of the implementation has been changed in the version 6.11 of `nicematrix` in order to allow the use of commands such as `\hline` or `\hdottedline` with the key `light-syntax`).

```
3153 \tl_clear_new:N \l_@@_new_body_tl
3154 \int_zero_new:N \l_@@_nb_cols_int
```

First, we treat the first row.

```
3155 \seq_pop_left:NN \l_@@_rows_seq \l_tmpa_tl
3156 \@@_line_with_light_syntax:V \l_tmpa_tl
```

Now, the other rows (with the same treatment, excepted that we have to insert `\\` between the rows).

```
3157 \seq_map_inline:Nn \l_@@_rows_seq
3158 {
3159   \tl_put_right:Nn \l_@@_new_body_tl { \\ }
3160   \@@_line_with_light_syntax:n { ##1 }
3161 }
3162 \int_compare:nNnT \l_@@_last_col_int = { -1 }
3163 {
3164   \int_set:Nn \l_@@_last_col_int
3165   { \l_@@_nb_cols_int - 1 + \l_@@_first_col_int }
3166 }
```

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.

```
3167 \@@_transform_preamble:
```

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

```
3168 \@@_array:V \g_@@_preamble_tl \l_@@_new_body_tl
3169 }
3170 \cs_new_protected:Npn \@@_line_with_light_syntax:n #1
3171 {
3172   \seq_clear_new:N \l_@@_cells_seq
3173   \seq_set_split:Nnn \l_@@_cells_seq { ~ } { #1 }
3174   \int_set:Nn \l_@@_nb_cols_int
3175   {
3176     \int_max:nn
3177     \l_@@_nb_cols_int
3178     { \seq_count:N \l_@@_cells_seq }
3179   }
3180   \seq_pop_left:NN \l_@@_cells_seq \l_tmpa_tl
3181   \tl_put_right:NV \l_@@_new_body_tl \l_tmpa_tl
3182   \seq_map_inline:Nn \l_@@_cells_seq
3183   { \tl_put_right:Nn \l_@@_new_body_tl { & ##1 } }
3184 }
3185 \cs_generate_variant:Nn \@@_line_with_light_syntax:n { V }
```

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`.

```
3186 \cs_new_protected:Npn \@@_analyze_end:Nn #1 #2
3187 {
3188   \str_if_eq:VnT \g_@@_name_env_str { #2 }
3189   { \@@_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.

```
3190   \end { #2 }
3191 }
```

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).

```
3192 \cs_new:Npn \@@_create_col_nodes:
3193 {
3194   \crrc
3195   \int_compare:nNnT \l_@@_first_col_int = 0
3196   {
3197     \omit
3198     \hbox_overlap_left:n
3199     {
3200       \bool_if:NT \l_@@_code_before_bool
3201         { \pgfsys@markposition { \@@_env: - col - 0 } }
3202       \pgfpicture
3203       \pgfrememberpicturepositiononpagetrue
3204       \pgfcoordinate { \@@_env: - col - 0 } \pgfpointorigin
3205       \str_if_empty:NF \l_@@_name_str
3206         { \pgfnodealias { \l_@@_name_str - col - 0 } { \@@_env: - col - 0 } }
3207       \endpgfpicture
3208       \skip_horizontal:N 2\col@sep
3209       \skip_horizontal:N \g_@@_width_first_col_dim
3210     }
3211     &
3212   }
3213   \omit
```

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

```
3214   \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`).

```
3215   \int_compare:nNnTF \l_@@_first_col_int = 0
3216   {
3217     \bool_if:NT \l_@@_code_before_bool
3218     {
3219       \hbox
3220       {
3221         \skip_horizontal:N -0.5\arrayrulewidth
3222         \pgfsys@markposition { \@@_env: - col - 1 }
3223         \skip_horizontal:N 0.5\arrayrulewidth
3224       }
3225     }
3226     \pgfpicture
3227     \pgfrememberpicturepositiononpagetrue
3228     \pgfcoordinate { \@@_env: - col - 1 }
3229     { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3230     \str_if_empty:NF \l_@@_name_str
3231     { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
3232     \endpgfpicture
3233   }
3234   {
3235     \bool_if:NT \l_@@_code_before_bool
3236     {
3237       \hbox
3238       {
3239         \skip_horizontal:N 0.5\arrayrulewidth
3240         \pgfsys@markposition { \@@_env: - col - 1 }
3241         \skip_horizontal:N -0.5\arrayrulewidth
```

```

3242     }
3243   }
3244   \pgfpicture
3245   \pgfrememberpicturepositiononpagetrue
3246   \pgfcoordinate { \@@_env: - col - 1 }
3247   { \pgfpoint { 0.5 \arrayrulewidth } \c_zero_dim }
3248   \str_if_empty:NF \l_@@_name_str
3249   { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
3250   \endpgfpicture
3251 }

```

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 this 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 it will just after be erased by a fixed value in the concerned cases.

```

3252   \skip_gset:Nn \g_tmpa_skip { 0 pt+plus 1 fill }
3253   \bool_if:NF \l_@@_auto_columns_width_bool
3254   { \dim_compare:nNnT \l_@@_columns_width_dim > \c_zero_dim }
3255   {
3256     \bool_lazy_and:nnTF
3257     \l_@@_auto_columns_width_bool
3258     { \bool_not_p:n \l_@@_block_auto_columns_width_bool }
3259     { \skip_gset_eq:NN \g_tmpa_skip \g_@@_max_cell_width_dim }
3260     { \skip_gset_eq:NN \g_tmpa_skip \l_@@_columns_width_dim }
3261     \skip_gadd:Nn \g_tmpa_skip { 2 \col@sep }
3262   }
3263   \skip_horizontal:N \g_tmpa_skip
3264   \hbox
3265   {
3266     \bool_if:NT \l_@@_code_before_bool
3267     {
3268       \hbox
3269       {
3270         \skip_horizontal:N -0.5\arrayrulewidth
3271         \pgfsys@markposition { \@@_env: - col - 2 }
3272         \skip_horizontal:N 0.5\arrayrulewidth
3273       }
3274     }
3275     \pgfpicture
3276     \pgfrememberpicturepositiononpagetrue
3277     \pgfcoordinate { \@@_env: - col - 2 }
3278     { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3279     \str_if_empty:NF \l_@@_name_str
3280     { \pgfnodealias { \l_@@_name_str - col - 2 } { \@@_env: - col - 2 } }
3281     \endpgfpicture
3282   }

```

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.

```

3283   \int_gset:Nn \g_tmpa_int 1
3284   \bool_if:NTF \g_@@_last_col_found_bool
3285   { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 3 } 0 } }
3286   { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 2 } 0 } }
3287   {
3288     &
3289     \omit
3290     \int_gincr:N \g_tmpa_int

```

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

```

3291     \skip_horizontal:N \g_tmpa_skip
3292     \bool_if:NT \l_@@_code_before_bool
3293     {

```

```

3294     \hbox
3295     {
3296         \skip_horizontal:N -0.5\arrayrulewidth
3297         \pgfsys@markposition
3298         { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3299         \skip_horizontal:N 0.5\arrayrulewidth
3300     }
3301 }

```

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

```

3302     \pgfpicture
3303     \pgfrememberpicturepositiononpagetrue
3304     \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3305     { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3306     \str_if_empty:NF \l_@@_name_str
3307     {
3308         \pgfnodealias
3309         { \l_@@_name_str - col - \int_eval:n { \g_tmpa_int + 1 } }
3310         { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3311     }
3312     \endpgfpicture
3313 }

3314 &
3315 \omit

```

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

```

3316     \int_compare:nNnT \g_@@_col_total_int = 1
3317     { \skip_gset:Nn \g_tmpa_skip { 0 pt~plus 1 fill } }
3318     \skip_horizontal:N \g_tmpa_skip
3319     \int_gincr:N \g_tmpa_int
3320     \bool_lazy_all:nT
3321     {
3322         { \bool_not_p:n \g_@@_delims_bool }
3323         { \bool_not_p:n \l_@@_tabular_bool }
3324         { \clist_if_empty_p:N \l_@@_vlines_clist }
3325         { \bool_not_p:n \l_@@_exterior_arraycolsep_bool }
3326         { ! \l_@@_bar_at_end_of_pream_bool }
3327     }
3328     { \skip_horizontal:N -\col@sep }
3329     \bool_if:NT \l_@@_code_before_bool
3330     {
3331         \hbox
3332         {
3333             \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.

```

3334         \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3335         { \skip_horizontal:N -\arraycolsep }
3336         \pgfsys@markposition
3337         { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3338         \skip_horizontal:N 0.5\arrayrulewidth
3339         \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3340         { \skip_horizontal:N \arraycolsep }
3341     }
3342 }
3343 \pgfpicture
3344     \pgfrememberpicturepositiononpagetrue
3345     \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3346     {
3347         \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool

```

```

3348         {
3349             \pgfpoint
3350             { - 0.5 \arrayrulewidth - \arraycolsep }
3351             \c_zero_dim
3352         }
3353         { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3354     }
3355     \str_if_empty:NF \l_@@_name_str
3356     {
3357         \pgfnodealias
3358         { \l_@@_name_str - col - \int_eval:n { \g_tmpa_int + 1 } }
3359         { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3360     }
3361     \endpgfpicture

3362     \bool_if:NT \g_@@_last_col_found_bool
3363     {
3364         \hbox_overlap_right:n
3365         {
3366             \skip_horizontal:N \g_@@_width_last_col_dim
3367             \bool_if:NT \l_@@_code_before_bool
3368             {
3369                 \pgfsys@markposition
3370                 { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3371             }
3372             \pgfpicture
3373             \pgfrememberpicturerepositiononpagetrue
3374             \pgfcoordinate
3375             { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3376             \pgfpointorigin
3377             \str_if_empty:NF \l_@@_name_str
3378             {
3379                 \pgfnodealias
3380                 {
3381                     \l_@@_name_str - col
3382                     - \int_eval:n { \g_@@_col_total_int + 1 }
3383                 }
3384                 { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3385             }
3386             \endpgfpicture
3387         }
3388     }
3389     \cr
3390 }

```

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

```

3391 \tl_const:Nn \c_@@_preamble_first_col_tl
3392 {
3393     >
3394     {

```

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).

```

3395     \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
3396     \bool_gset_true:N \g_@@_after_col_zero_bool
3397     \@@_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.

```

3398     \hbox_set:Nw \l_@@_cell_box
3399     \@@_math_toggle_token:
3400     \bool_if:NT \l_@@_small_bool \scriptstyle

```

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".

```

3401     \bool_lazy_and:nnT
3402     { \int_compare_p:nNn \c@iRow > 0 }
3403     {
3404         \bool_lazy_or_p:nn
3405         { \int_compare_p:nNn \l_@@_last_row_int < 0 }
3406         { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
3407     }
3408     {
3409         \l_@@_code_for_first_col_tl
3410         \xglobal \colorlet { nicematrix-first-col } { . }
3411     }
3412 }

```

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`.

```

3413     l
3414     <
3415     {
3416         \@@_math_toggle_token:
3417         \hbox_set_end:
3418         \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
3419         \@@_adjust_size_box:
3420         \@@_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.

```

3421     \dim_gset:Nn \g_@@_width_first_col_dim
3422     { \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.

```

3423     \hbox_overlap_left:n
3424     {
3425         \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > \c_zero_dim
3426         \@@_node_for_cell:
3427         { \box_use_drop:N \l_@@_cell_box }
3428         \skip_horizontal:N \l_@@_left_delim_dim
3429         \skip_horizontal:N \l_@@_left_margin_dim
3430         \skip_horizontal:N \l_@@_extra_left_margin_dim
3431     }
3432     \bool_gset_false:N \g_@@_empty_cell_bool
3433     \skip_horizontal:N -2\col@sep
3434 }
3435 }

```

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

```

3436 \tl_const:Nn \c_@@_preamble_last_col_tl
3437 {
3438     >
3439     {
3440         \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).

```

3441     \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.

```

3442     \bool_gset_true:N \g_@@_last_col_found_bool
3443     \int_gincr:N \c@jCol
3444     \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.

```

3445     \hbox_set:Nw \l_@@_cell_box
3446     \@@_math_toggle_token:

```

```
3447 \bool_if:NT \l_@@_small_bool \scriptstyle
```

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".

```
3448 \int_compare:nNnT \c@iRow > 0
3449 {
3450   \bool_lazy_or:nnT
3451     { \int_compare_p:nNn \l_@@_last_row_int < 0 }
3452     { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
3453   {
3454     \l_@@_code_for_last_col_tl
3455     \xglobal \colorlet { nicematrix-last-col } { . }
3456   }
3457 }
3458 }
3459 l
3460 <
3461 {
3462   \@@_math_toggle_token:
3463   \hbox_set_end:
3464   \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
3465   \@@_adjust_size_box:
3466   \@@_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.

```
3467 \dim_gset:Nn \g_@@_width_last_col_dim
3468   { \dim_max:nn \g_@@_width_last_col_dim { \box_wd:N \l_@@_cell_box } }
3469 \skip_horizontal:N -2\col@sep
```

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

```
3470 \hbox_overlap_right:n
3471 {
3472   \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \c_zero_dim
3473   {
3474     \skip_horizontal:N \l_@@_right_delim_dim
3475     \skip_horizontal:N \l_@@_right_margin_dim
3476     \skip_horizontal:N \l_@@_extra_right_margin_dim
3477     \@@_node_for_cell:
3478   }
3479 }
3480 \bool_gset_false:N \g_@@_empty_cell_bool
3481 }
3482 }
```

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

```
3483 \NewDocumentEnvironment { NiceArray } { }
3484 {
3485   \bool_gset_false:N \g_@@_delims_bool
3486   \str_if_empty:NT \g_@@_name_env_str
3487     { \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).

```
3488 \NiceArrayWithDelims . .
3489 }
3490 { \endNiceArrayWithDelims }
```

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

```
3491 \cs_new_protected:Npn \@@_def_env:nnn #1 #2 #3
3492 {
3493   \NewDocumentEnvironment { #1 NiceArray } { }
```

```

3494     {
3495       \bool_gset_true:N \g_@@_delims_bool
3496       \str_if_empty:NT \g_@@_name_env_str
3497         { \str_gset:Nn \g_@@_name_env_str { #1 NiceArray } }
3498       \@@_test_if_math_mode:
3499       \NiceArrayWithDelims #2 #3
3500     }
3501     { \endNiceArrayWithDelims }
3502   }

3503 \@@_def_env:nnn p ( )
3504 \@@_def_env:nnn b [ ]
3505 \@@_def_env:nnn B \{ \}
3506 \@@_def_env:nnn v | |
3507 \@@_def_env:nnn V \| \|

```

14 The environment `{NiceMatrix}` and its variants

```

3508 \cs_new_protected:Npn \@@_begin_of_NiceMatrix:nn #1 #2
3509   {
3510     \bool_set_false:N \l_@@_preamble_bool
3511     \tl_clear:N \l_tmpa_tl
3512     \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3513       { \tl_set:Nn \l_tmpa_tl { @ { } } }
3514     \tl_put_right:Nn \l_tmpa_tl
3515       {
3516         *
3517         {
3518           \int_case:nnF \l_@@_last_col_int
3519             {
3520               { -2 } { \c@MaxMatrixCols }
3521               { -1 } { \int_eval:n { \c@MaxMatrixCols + 1 } }
3522             }
3523             { \int_eval:n { \l_@@_last_col_int - 1 } }
3524           }
3525           { #2 }
3526         }
3527     \tl_set:Nn \l_tmpb_tl { \use:c { #1 NiceArray } }
3528     \exp_args:NV \l_tmpb_tl \l_tmpa_tl
3529   }

3530 \cs_generate_variant:Nn \@@_begin_of_NiceMatrix:nn { n V }

3531 \clist_map_inline:nn { p , b , B , v , V }
3532   {
3533     \NewDocumentEnvironment { #1 NiceMatrix } { ! O { } }
3534     {
3535       \bool_gset_true:N \g_@@_delims_bool
3536       \str_gset:Nn \g_@@_name_env_str { #1 NiceMatrix }
3537       \keys_set:nn { NiceMatrix / NiceMatrix } { ##1 }
3538       \@@_begin_of_NiceMatrix:nV { #1 } \l_@@_columns_type_tl
3539     }
3540     { \use:c { end #1 NiceArray } }
3541   }

```

We define also an environment `{NiceMatrix}`

```

3542 \NewDocumentEnvironment { NiceMatrix } { ! O { } }
3543   {
3544     \str_gset:Nn \g_@@_name_env_str { NiceMatrix }

```



```

3545 \keys_set:nn { NiceMatrix / NiceMatrix } { #1 }
3546 \bool_lazy_or:nnT
3547   { \clist_if_empty_p:N \l_@@_vlines_clist }
3548   { \l_@@_except_borders_bool }
3549   { \bool_set_true:N \l_@@_NiceMatrix_without_vlines_bool }
3550 \@@_begin_of_NiceMatrix:nV { } \l_@@_columns_type_tl
3551 }
3552 { \endNiceArray }

```

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

```

3553 \cs_new_protected:Npn \@@_NotEmpty:
3554   { \bool_gset_true:N \g_@@_not_empty_cell_bool }

```

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

```

3555 \NewDocumentEnvironment { NiceTabular } { 0 { } m ! 0 { } }
3556   {

```

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`.

```

3557   \dim_compare:nNnT \l_@@_width_dim = \c_zero_dim
3558     { \dim_set_eq:NN \l_@@_width_dim \linewidth }
3559   \str_gset:Nn \g_@@_name_env_str { NiceTabular }
3560   \keys_set:nn { NiceMatrix / NiceTabular } { #1 , #3 }
3561   \tl_if_empty:NF \l_@@_short_caption_tl
3562     {
3563       \tl_if_empty:NT \l_@@_caption_tl
3564         {
3565           \@@_error_or_warning:n { short-caption~without~caption }
3566           \tl_set_eq:NN \l_@@_caption_tl \l_@@_short_caption_tl
3567         }
3568       }
3569   \tl_if_empty:NF \l_@@_label_tl
3570     {
3571       \tl_if_empty:NT \l_@@_caption_tl
3572         { \@@_error_or_warning:n { label~without~caption } }
3573     }
3574   \NewDocumentEnvironment { TabularNote } { b }
3575     {
3576       \bool_if:NTF \l_@@_in_code_after_bool
3577         { \@@_error_or_warning:n { TabularNote~in~CodeAfter } }
3578         {
3579           \tl_if_empty:NF \g_@@_tabularnote_tl
3580             { \tl_gput_right:Nn \g_@@_tabularnote_tl { \par } }
3581             \tl_gput_right:Nn \g_@@_tabularnote_tl { ##1 }
3582         }
3583     }
3584     { }
3585   \bool_set_true:N \l_@@_tabular_bool
3586   \NiceArray { #2 }
3587 }
3588 { \endNiceArray }

3589 \cs_set_protected:Npn \@@_newcolumn_type #1
3590   {
3591     \cs_if_free:cT { NC @ find @ #1 }
3592     { \NC@list \expandafter { \the \NC@list \NC@do #1 } }
3593     \cs_set:cpn { NC @ find @ #1 } ##1 #1 { \NC@ { ##1 } }
3594     \peek_meaning:NTF [
3595       { \newcol@ #1 }
3596       { \newcol@ #1 [ 0 ] }
3597   }

```

```

3598 \NewDocumentEnvironment { NiceTabularX } { m O { } m ! O { } }
3599 {

```

The following code prevents the expansion of the ‘X’ columns with the definition of that columns in `tabularx` (this would result in an error in `{NiceTabularX}`).

```

3600 \IfPackageLoadedTF { tabularx }
3601   { \newcolumntype { X } { \@@_X } }
3602   { }
3603 \str_gset:Nn \g_@@_name_env_str { NiceTabularX }
3604 \dim_zero_new:N \l_@@_width_dim
3605 \dim_set:Nn \l_@@_width_dim { #1 }
3606 \keys_set:nn { NiceMatrix / NiceTabular } { #2 , #4 }
3607 \bool_set_true:N \l_@@_tabular_bool
3608 \NiceArray { #3 }
3609 }
3610 { \endNiceArray }

```

```

3611 \NewDocumentEnvironment { NiceTabular* } { m O { } m ! O { } }
3612 {
3613   \str_gset:Nn \g_@@_name_env_str { NiceTabular* }
3614   \dim_set:Nn \l_@@_tabular_width_dim { #1 }
3615   \keys_set:nn { NiceMatrix / NiceTabular } { #2 , #4 }
3616   \bool_set_true:N \l_@@_tabular_bool
3617   \NiceArray { #3 }
3618 }
3619 { \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`).

```

3620 \cs_new_protected:Npn \@@_deal_with_rounded_corners:
3621 {
3622   \bool_lazy_all:nT
3623   {
3624     { \int_compare_p:nNn \l_@@_tab_rounded_corners_dim > \c_zero_dim }
3625     \l_@@_hvlines_bool
3626     { ! \g_@@_delims_bool }
3627     { ! \l_@@_except_borders_bool }
3628   }
3629   {
3630     \bool_set_true:N \l_@@_except_borders_bool
3631     \clist_if_empty:NF \l_@@_corners_clist
3632     { \@@_error:n { hvlines,~rounded-corners~and~corners } }
3633     \tl_gput_right:Nn \g_@@_pre_code_after_tl
3634     {
3635       \@@_stroke_block:nnn
3636       {
3637         rounded-corners = \dim_use:N \l_@@_tab_rounded_corners_dim ,
3638         draw = \l_@@_rules_color_tl
3639       }
3640       { 1-1 }
3641       { \int_use:N \c@iRow - \int_use:N \c@jCol }
3642     }
3643   }
3644 }
3645 %
3646 %
3647 % \medskip

```

```

3648 % \begin{macrocode}
3649 \cs_new_protected:Npn \@@_after_array:
3650 {
3651 \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.

```

3652 \bool_if:NT \g_@@_last_col_found_bool
3653 { \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`.

```

3654 \bool_if:NT \l_@@_last_col_without_value_bool
3655 { \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.

```

3656 \bool_if:NT \l_@@_last_row_without_value_bool
3657 { \int_set_eq:NN \l_@@_last_row_int \g_@@_row_total_int }

```

```

3658 \tl_gput_right:Nx \g_@@_aux_tl
3659 {
3660 \seq_gset_from_clist:Nn \exp_not:N \g_@@_size_seq
3661 {
3662 \int_use:N \l_@@_first_row_int ,
3663 \int_use:N \c@iRow ,
3664 \int_use:N \g_@@_row_total_int ,
3665 \int_use:N \l_@@_first_col_int ,
3666 \int_use:N \c@jCol ,
3667 \int_use:N \g_@@_col_total_int
3668 }
3669 }

```

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`).

```

3670 \seq_if_empty:NF \g_@@_pos_of_blocks_seq
3671 {
3672 \tl_gput_right:Nx \g_@@_aux_tl
3673 {
3674 \seq_gset_from_clist:Nn \exp_not:N \g_@@_pos_of_blocks_seq
3675 { \seq_use:Nnnn \g_@@_pos_of_blocks_seq , , , }
3676 }
3677 }
3678 \seq_if_empty:NF \g_@@_multicolumn_cells_seq
3679 {
3680 \tl_gput_right:Nx \g_@@_aux_tl
3681 {
3682 \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_cells_seq
3683 { \seq_use:Nnnn \g_@@_multicolumn_cells_seq , , , }
3684 \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_sizes_seq
3685 { \seq_use:Nnnn \g_@@_multicolumn_sizes_seq , , , }
3686 }
3687 }

```

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

```

3688 \@@_create_diag_nodes:

```

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

```

3689 \pgfpicture
3690 \int_step_inline:nn \c@iRow
3691 {
3692   \pgfnodealias
3693     { \@@_env: - ##1 - last }
3694     { \@@_env: - ##1 - \int_use:N \c@jCol }
3695 }
3696 \int_step_inline:nn \c@jCol
3697 {
3698   \pgfnodealias
3699     { \@@_env: - last - ##1 }
3700     { \@@_env: - \int_use:N \c@iRow - ##1 }
3701 }
3702 \str_if_empty:NF \l_@@_name_str
3703 {
3704   \int_step_inline:nn \c@iRow
3705   {
3706     \pgfnodealias
3707       { \l_@@_name_str - ##1 - last }
3708       { \@@_env: - ##1 - \int_use:N \c@jCol }
3709   }
3710   \int_step_inline:nn \c@jCol
3711   {
3712     \pgfnodealias
3713       { \l_@@_name_str - last - ##1 }
3714       { \@@_env: - \int_use:N \c@iRow - ##1 }
3715   }
3716 }
3717 \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.

```

3718 \bool_if:NT \l_@@_parallelize_diags_bool
3719 {
3720   \int_gzero_new:N \g_@@_ddots_int
3721   \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.

```

3722   \dim_gzero_new:N \g_@@_delta_x_one_dim
3723   \dim_gzero_new:N \g_@@_delta_y_one_dim
3724   \dim_gzero_new:N \g_@@_delta_x_two_dim
3725   \dim_gzero_new:N \g_@@_delta_y_two_dim
3726 }
3727 \int_zero_new:N \l_@@_initial_i_int
3728 \int_zero_new:N \l_@@_initial_j_int
3729 \int_zero_new:N \l_@@_final_i_int
3730 \int_zero_new:N \l_@@_final_j_int
3731 \bool_set_false:N \l_@@_initial_open_bool
3732 \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.

```

3733 \bool_if:NT \l_@@_small_bool
3734 {

```

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

```

3735     \dim_set:Nn \l_@@_xdots_radius_dim { 0.7 \l_@@_xdots_radius_dim }
3736     \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.

```

3737     \dim_set:Nn \l_@@_xdots_shorten_start_dim
3738     { 0.6 \l_@@_xdots_shorten_start_dim }
3739     \dim_set:Nn \l_@@_xdots_shorten_end_dim
3740     { 0.6 \l_@@_xdots_shorten_end_dim }
3741 }

```

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

```

3742 \@@_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.

```

3743 \@@_compute_corners:

```

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

```

3744 \@@_adjust_pos_of_blocks_seq:
3745 \@@_deal_with_rounded_corners:
3746 \tl_if_empty:NF \l_@@_hlines_clist \@@_draw_hlines:
3747 \tl_if_empty:NF \l_@@_vlines_clist \@@_draw_vlines:

```

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

```

3748 \IfPackageLoadedTF { tikz }
3749 {
3750     \tikzset
3751     {
3752         every~picture / .style =
3753         {
3754             overlay ,
3755             remember~picture ,
3756             name~prefix = \@@_env: -
3757         }
3758     }
3759 }
3760 { }
3761 \cs_set_eq:NN \ialign \@@_old_ialign:
3762 \cs_set_eq:NN \SubMatrix \@@_SubMatrix
3763 \cs_set_eq:NN \UnderBrace \@@_UnderBrace
3764 \cs_set_eq:NN \OverBrace \@@_OverBrace
3765 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
3766 \cs_set_eq:NN \line \@@_line
3767 \g_@@_pre_code_after_tl
3768 \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.

```

3769 \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).

```

3770 \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`).

```
3771 \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
3772 { \@@_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:`.

```
3773 \bool_set_true:N \l_@@_in_code_after_bool
3774 \exp_last_unbraced:NV \@@_CodeAfter_keys: \g_nicematrix_code_after_tl
3775 \scan_stop:
3776 \tl_gclear:N \g_nicematrix_code_after_tl
3777 \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.

```
3778 \tl_if_empty:NF \g_@@_pre_code_before_tl
3779 {
3780   \tl_gput_right:Nx \g_@@_aux_tl
3781   {
3782     \tl_gset:Nn \exp_not:N \g_@@_pre_code_before_tl
3783     { \exp_not:V \g_@@_pre_code_before_tl }
3784   }
3785   \tl_gclear:N \g_@@_pre_code_before_tl
3786 }
3787 \tl_if_empty:NF \g_nicematrix_code_before_tl
3788 {
3789   \tl_gput_right:Nx \g_@@_aux_tl
3790   {
3791     \tl_gset:Nn \exp_not:N \g_@@_code_before_tl
3792     { \exp_not:V \g_nicematrix_code_before_tl }
3793   }
3794   \tl_gclear:N \g_nicematrix_code_before_tl
3795 }

3796 \str_gclear:N \g_@@_name_env_str
3797 \@@_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`.

```
3798 \cs_gset_eq:NN \CT@arc@ \@@_old_CT@arc@
3799 }
```

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`.

```
3800 \NewDocumentCommand \@@_CodeAfter_keys: { 0 { } }
3801 { \keys_set:nn { NiceMatrix / CodeAfter } { #1 } }
```

We remind that the first mandatory argument of the command `\Block` is the size of the block with the special format $i-j$. However, the user is allowed to omit i or j (or both). This will be interpreted as: the last row (resp. column) of the block will be the last row (resp. column) of the block (without the potential exterior row—resp. column—of the array). By convention, this is stored in `\g_@@_pos_of_blocks_seq` (and `\g_@@_blocks_seq`) as a number of rows (resp. columns) for the

¹²e.g. `\color[rgb]{0.5,0.5,0}`

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).

```

3802 \cs_new_protected:Npn \@@_adjust_pos_of_blocks_seq:
3803   {
3804     \seq_gset_map_x:NNn \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq
3805     { \@@_adjust_pos_of_blocks_seq_i:nnnnn ##1 }
3806   }

```

The following command must *not* be protected.

```

3807 \cs_new:Npn \@@_adjust_pos_of_blocks_seq_i:nnnnn #1 #2 #3 #4 #5
3808   {
3809     { #1 }
3810     { #2 }
3811     {
3812       \int_compare:nNnTF { #3 } > { 99 }
3813       { \int_use:N \c@iRow }
3814       { #3 }
3815     }
3816     {
3817       \int_compare:nNnTF { #4 } > { 99 }
3818       { \int_use:N \c@jCol }
3819       { #4 }
3820     }
3821     { #5 }
3822   }

```

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).

```

3823 \hook_gput_code:nnn { begindocument } { . }
3824   {
3825     \cs_new_protected:Npx \@@_draw_dotted_lines:
3826     {
3827       \c_@@_pgfortikzpicture_tl
3828       \@@_draw_dotted_lines_i:
3829       \c_@@_endpgfortikzpicture_tl
3830     }
3831   }

```

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

```

3832 \cs_new_protected:Npn \@@_draw_dotted_lines_i:
3833   {
3834     \pgfrememberpicturepositiononpagetrue
3835     \pgf@relevantforpicturesizefalse
3836     \g_@@_HVdotsfor_lines_tl
3837     \g_@@_Vdots_lines_tl
3838     \g_@@_Ddots_lines_tl
3839     \g_@@_Iddots_lines_tl
3840     \g_@@_Cdots_lines_tl
3841     \g_@@_Ldots_lines_tl
3842   }

3843 \cs_new_protected:Npn \@@_restore_iRow_jCol:
3844   {
3845     \cs_if_exist:NT \theiRow { \int_gset_eq:NN \c@iRow \l_@@_old_iRow_int }
3846     \cs_if_exist:NT \thejCol { \int_gset_eq:NN \c@jCol \l_@@_old_jCol_int }
3847   }

```

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

```

3848 \pgfdeclareshape { @@_diag_node }
3849 {
3850   \savedanchor { \five }
3851   {
3852     \dim_gset_eq:NN \pgf@x \l_tmpa_dim
3853     \dim_gset_eq:NN \pgf@y \l_tmpb_dim
3854   }
3855   \anchor { 5 } { \five }
3856   \anchor { center } { \pgfpointorigin }
3857 }

```

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).

```

3858 \cs_new_protected:Npn @@_create_diag_nodes:
3859 {
3860   \pgfpicture
3861   \pgfrememberpicturepositiononpagetrue
3862   \int_step_inline:nn { \int_max:nn \c@iRow \c@jCol }
3863   {
3864     @@_qpoint:n { col - \int_min:nn { ##1 } { \c@jCol + 1 } }
3865     \dim_set_eq:NN \l_tmpa_dim \pgf@x
3866     @@_qpoint:n { row - \int_min:nn { ##1 } { \c@iRow + 1 } }
3867     \dim_set_eq:NN \l_tmpb_dim \pgf@y
3868     @@_qpoint:n { col - \int_min:nn { ##1 + 1 } { \c@jCol + 1 } }
3869     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
3870     @@_qpoint:n { row - \int_min:nn { ##1 + 1 } { \c@iRow + 1 } }
3871     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
3872     \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.

```

3873     \dim_set:Nn \l_tmpa_dim { ( \l_@@_tmpc_dim - \l_tmpa_dim ) / 2 }
3874     \dim_set:Nn \l_tmpb_dim { ( \l_@@_tmpd_dim - \l_tmpb_dim ) / 2 }
3875     \pgfnode { @@_diag_node } { center } { } { \@@_env: - ##1 } { }
3876     \str_if_empty:NF \l_@@_name_str
3877     { \pgfnodealias { \l_@@_name_str - ##1 } { \@@_env: - ##1 } }
3878   }

```

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

```

3879   \int_set:Nn \l_tmpa_int { \int_max:nn \c@iRow \c@jCol + 1 }
3880   @@_qpoint:n { row - \int_min:nn { \l_tmpa_int } { \c@iRow + 1 } }
3881   \dim_set_eq:NN \l_tmpa_dim \pgf@y
3882   @@_qpoint:n { col - \int_min:nn { \l_tmpa_int } { \c@jCol + 1 } }
3883   \pgfcoordinate
3884   { \@@_env: - \int_use:N \l_tmpa_int } { \pgfpoint \pgf@x \l_tmpa_dim }
3885   \pgfnodealias
3886   { \@@_env: - last }
3887   { \@@_env: - \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
3888   \str_if_empty:NF \l_@@_name_str
3889   {
3890     \pgfnodealias
3891     { \l_@@_name_str - \int_use:N \l_tmpa_int }
3892     { \@@_env: - \int_use:N \l_tmpa_int }
3893     \pgfnodealias
3894     { \l_@@_name_str - last }
3895     { \@@_env: - last }
3896   }
3897   \endpgfpicture
3898 }

```


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.

$$\left(\begin{array}{ccccc} a + b + c & a + b & & a & \\ & a & \dots & \dots & \dots \\ & a & & a + b & a + b + c \end{array} \right)$$

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
3900   {
```

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

```
3901   \cs_set:cpn { @@ _ dotted _ #1 - #2 } { }
```

Initialization of variables.

```
3902   \int_set:Nn \l_@@_initial_i_int { #1 }
3903   \int_set:Nn \l_@@_initial_j_int { #2 }
3904   \int_set:Nn \l_@@_final_i_int { #1 }
3905   \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.

```
3906   \bool_set_false:N \l_@@_stop_loop_bool
3907   \bool_do_until:Nn \l_@@_stop_loop_bool
3908   {
3909     \int_add:Nn \l_@@_final_i_int { #3 }
3910     \int_add:Nn \l_@@_final_j_int { #4 }
```

We test if we are still in the matrix.

```
3911   \bool_set_false:N \l_@@_final_open_bool
3912   \int_compare:nNnTF \l_@@_final_i_int > \l_@@_row_max_int
3913   {
3914     \int_compare:nNnTF { #3 } = 1
3915     { \bool_set_true:N \l_@@_final_open_bool }
3916     {
3917       \int_compare:nNnT \l_@@_final_j_int > \l_@@_col_max_int
3918       { \bool_set_true:N \l_@@_final_open_bool }
3919     }
3920   }
3921   {
3922     \int_compare:nNnTF \l_@@_final_j_int < \l_@@_col_min_int
```

```

3923     {
3924     \int_compare:nNnT { #4 } = { -1 }
3925     { \bool_set_true:N \l_@@_final_open_bool }
3926     }
3927     {
3928     \int_compare:nNnT \l_@@_final_j_int > \l_@@_col_max_int
3929     {
3930     \int_compare:nNnT { #4 } = 1
3931     { \bool_set_true:N \l_@@_final_open_bool }
3932     }
3933     }
3934     }
3935     \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.

```

3936     {
We do a step backwards.
3937     \int_sub:Nn \l_@@_final_i_int { #3 }
3938     \int_sub:Nn \l_@@_final_j_int { #4 }
3939     \bool_set_true:N \l_@@_stop_loop_bool
3940     }

```

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`.

```

3941     {
3942     \cs_if_exist:cTF
3943     {
3944     @@ _ dotted _
3945     \int_use:N \l_@@_final_i_int -
3946     \int_use:N \l_@@_final_j_int
3947     }
3948     {
3949     \int_sub:Nn \l_@@_final_i_int { #3 }
3950     \int_sub:Nn \l_@@_final_j_int { #4 }
3951     \bool_set_true:N \l_@@_final_open_bool
3952     \bool_set_true:N \l_@@_stop_loop_bool
3953     }
3954     {
3955     \cs_if_exist:cTF
3956     {
3957     pgf @ sh @ ns @ \@@_env:
3958     - \int_use:N \l_@@_final_i_int
3959     - \int_use:N \l_@@_final_j_int
3960     }
3961     { \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.

```

3962     {
3963     \cs_set:cpn
3964     {
3965     @@ _ dotted _
3966     \int_use:N \l_@@_final_i_int -
3967     \int_use:N \l_@@_final_j_int
3968     }
3969     { }
3970     }
3971     }
3972     }
3973     }

```

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

```

3974 \bool_set_false:N \l_@@_stop_loop_bool
3975 \bool_do_until:Nn \l_@@_stop_loop_bool
3976 {
3977   \int_sub:Nn \l_@@_initial_i_int { #3 }
3978   \int_sub:Nn \l_@@_initial_j_int { #4 }
3979   \bool_set_false:N \l_@@_initial_open_bool
3980   \int_compare:nNnTF \l_@@_initial_i_int < \l_@@_row_min_int
3981     {
3982       \int_compare:nNnTF { #3 } = 1
3983         { \bool_set_true:N \l_@@_initial_open_bool }
3984         {
3985           \int_compare:nNnT \l_@@_initial_j_int = { \l_@@_col_min_int -1 }
3986             { \bool_set_true:N \l_@@_initial_open_bool }
3987         }
3988     }
3989   {
3990     \int_compare:nNnTF \l_@@_initial_j_int < \l_@@_col_min_int
3991       {
3992         \int_compare:nNnT { #4 } = 1
3993           { \bool_set_true:N \l_@@_initial_open_bool }
3994       }
3995       {
3996         \int_compare:nNnT \l_@@_initial_j_int > \l_@@_col_max_int
3997           {
3998             \int_compare:nNnT { #4 } = { -1 }
3999               { \bool_set_true:N \l_@@_initial_open_bool }
4000           }
4001       }
4002   }
4003   \bool_if:NTF \l_@@_initial_open_bool
4004     {
4005       \int_add:Nn \l_@@_initial_i_int { #3 }
4006       \int_add:Nn \l_@@_initial_j_int { #4 }
4007       \bool_set_true:N \l_@@_stop_loop_bool
4008     }
4009     {
4010       \cs_if_exist:cTF
4011         {
4012           @@ _ dotted _
4013           \int_use:N \l_@@_initial_i_int -
4014           \int_use:N \l_@@_initial_j_int
4015         }
4016         {
4017           \int_add:Nn \l_@@_initial_i_int { #3 }
4018           \int_add:Nn \l_@@_initial_j_int { #4 }
4019           \bool_set_true:N \l_@@_initial_open_bool
4020           \bool_set_true:N \l_@@_stop_loop_bool
4021         }
4022         {
4023           \cs_if_exist:cTF
4024             {
4025               pgf @ sh @ ns @ \@@_env:
4026               - \int_use:N \l_@@_initial_i_int
4027               - \int_use:N \l_@@_initial_j_int
4028             }
4029             { \bool_set_true:N \l_@@_stop_loop_bool }
4030             {
4031               \cs_set:cpn
4032                 {
4033                   @@ _ dotted _
4034                   \int_use:N \l_@@_initial_i_int -

```

```

4035             \int_use:N \l_@@_initial_j_int
4036         }
4037     { }
4038 }
4039 }
4040 }
4041 }

```

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.

```

4042 \seq_gput_right:Nx \g_@@_pos_of_xdots_seq
4043 {
4044     { \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`.

```

4045     { \int_min:nn \l_@@_initial_j_int \l_@@_final_j_int }
4046     { \int_use:N \l_@@_final_i_int }
4047     { \int_max:nn \l_@@_initial_j_int \l_@@_final_j_int }
4048     { } % for the name of the block
4049 }
4050 }

```

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 wheter 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.

```

4051 \cs_new_protected:Npn \@@_open_shorten:
4052 {
4053     \bool_if:NT \l_@@_initial_open_bool
4054     { \dim_zero:N \l_@@_xdots_shorten_start_dim }
4055     \bool_if:NT \l_@@_final_open_bool
4056     { \dim_zero:N \l_@@_xdots_shorten_end_dim }
4057 }

```

The following commmand (*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).

```

4058 \cs_new_protected:Npn \@@_adjust_to_submatrix:nn #1 #2
4059 {
4060     \int_set:Nn \l_@@_row_min_int 1
4061     \int_set:Nn \l_@@_col_min_int 1
4062     \int_set_eq:NN \l_@@_row_max_int \c@iRow
4063     \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`.

```

4064 \seq_map_inline:Nn \g_@@_submatrix_seq
4065 { \@@_adjust_to_submatrix:nnnnnn { #1 } { #2 } ##1 }
4066 }

```

#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.

```

4067 \cs_set_protected:Npn \@@_adjust_to_submatrix:nnnnnn #1 #2 #3 #4 #5 #6
4068 {
4069     \bool_if:nT
4070     {
4071         \int_compare_p:n { #3 <= #1 }
4072         && \int_compare_p:n { #1 <= #5 }

```

```

4073     && \int_compare_p:n { #4 <= #2 }
4074     && \int_compare_p:n { #2 <= #6 }
4075   }
4076   {
4077     \int_set:Nn \l_@@_row_min_int { \int_max:nn \l_@@_row_min_int { #3 } }
4078     \int_set:Nn \l_@@_col_min_int { \int_max:nn \l_@@_col_min_int { #4 } }
4079     \int_set:Nn \l_@@_row_max_int { \int_min:nn \l_@@_row_max_int { #5 } }
4080     \int_set:Nn \l_@@_col_max_int { \int_min:nn \l_@@_col_max_int { #6 } }
4081   }
4082 }

4083 \cs_new_protected:Npn \@@_set_initial_coords:
4084 {
4085   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4086   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4087 }
4088 \cs_new_protected:Npn \@@_set_final_coords:
4089 {
4090   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4091   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4092 }
4093 \cs_new_protected:Npn \@@_set_initial_coords_from_anchor:n #1
4094 {
4095   \pgfpointanchor
4096   {
4097     \@@_env:
4098     - \int_use:N \l_@@_initial_i_int
4099     - \int_use:N \l_@@_initial_j_int
4100   }
4101   { #1 }
4102   \@@_set_initial_coords:
4103 }
4104 \cs_new_protected:Npn \@@_set_final_coords_from_anchor:n #1
4105 {
4106   \pgfpointanchor
4107   {
4108     \@@_env:
4109     - \int_use:N \l_@@_final_i_int
4110     - \int_use:N \l_@@_final_j_int
4111   }
4112   { #1 }
4113   \@@_set_final_coords:
4114 }
4115 \cs_new_protected:Npn \@@_open_x_initial_dim:
4116 {
4117   \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
4118   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
4119   {
4120     \cs_if_exist:cT
4121     { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4122     {
4123       \pgfpointanchor
4124       { \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4125       { west }
4126       \dim_set:Nn \l_@@_x_initial_dim
4127       { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
4128     }
4129   }

```

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

```

4130   \dim_compare:nNnT \l_@@_x_initial_dim = \c_max_dim
4131   {
4132     \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }

```

```

4133     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4134     \dim_add:Nn \l_@@_x_initial_dim \col@sep
4135   }
4136 }
4137 \cs_new_protected:Npn \@@_open_x_final_dim:
4138 {
4139   \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
4140   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
4141   {
4142     \cs_if_exist:cT
4143     { \pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4144     {
4145       \pgfpointanchor
4146       { \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4147       { east }
4148       \dim_set:Nn \l_@@_x_final_dim
4149       { \dim_max:nn \l_@@_x_final_dim \pgf@x }
4150     }
4151   }

```

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

```

4152   \dim_compare:nNnT \l_@@_x_final_dim = { - \c_max_dim }
4153   {
4154     \@@_qpoint:n { col - \int_eval:n { \l_@@_final_j_int + 1 } }
4155     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4156     \dim_sub:Nn \l_@@_x_final_dim \col@sep
4157   }
4158 }

```

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.

```

4159 \cs_new_protected:Npn \@@_draw_Ldots:nnn #1 #2 #3
4160 {
4161   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4162   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4163   {
4164     \@@_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.

```

4165   \group_begin:
4166   \@@_open_shorten:
4167   \int_compare:nNnTF { #1 } = 0
4168   { \color { nicematrix-first-row } }
4169   {

```

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.

```

4170     \int_compare:nNnT { #1 } = \l_@@_last_row_int
4171     { \color { nicematrix-last-row } }
4172   }
4173   \keys_set:nn { NiceMatrix / xdots } { #3 }
4174   \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4175   \@@_actually_draw_Ldots:
4176   \group_end:
4177 }
4178 }

```

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`.

```

4179 \cs_new_protected:Npn \@@_actually_draw_Ldots:
4180 {
4181   \bool_if:NTF \l_@@_initial_open_bool
4182     {
4183     \@@_open_x_initial_dim:
4184     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4185     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4186     }
4187     { \@@_set_initial_coords_from_anchor:n { base~east } }
4188   \bool_if:NTF \l_@@_final_open_bool
4189     {
4190     \@@_open_x_final_dim:
4191     \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4192     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4193     }
4194     { \@@_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.

```

4195   \bool_lazy_all:nTF
4196     {
4197     \l_@@_initial_open_bool
4198     \l_@@_final_open_bool
4199     { \int_compare_p:nNn \l_@@_initial_i_int = \l_@@_last_row_int }
4200     }
4201     {
4202     \dim_add:Nn \l_@@_y_initial_dim \c_@@_shift_Ldots_last_row_dim
4203     \dim_add:Nn \l_@@_y_final_dim \c_@@_shift_Ldots_last_row_dim
4204     }

```

We raise the line of a quantity equal to the radius of the dots because we want the dots really “on” the line of texte. Of course, maybe we should not do that when the option `line-style` is used (?).

```

4205   {
4206     \dim_add:Nn \l_@@_y_initial_dim \l_@@_xdots_radius_dim
4207     \dim_add:Nn \l_@@_y_final_dim \l_@@_xdots_radius_dim
4208   }
4209   \@@_draw_line:
4210 }

```

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.

```

4211 \cs_new_protected:Npn \@@_draw_Cdots:nnn #1 #2 #3
4212 {
4213   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4214   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4215   {
4216     \@@_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.

```

4217   \group_begin:
4218     \@@_open_shorten:
4219     \int_compare:nNnTF { #1 } = 0
4220     { \color { nicematrix-first-row } }
4221     {

```

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.

```

4222         \int_compare:nNnT { #1 } = \l_@@_last_row_int
4223         { \color { nicematrix-last-row } }
4224     }
4225     \keys_set:nn { NiceMatrix / xdots } { #3 }
4226     \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4227     \@@_actually_draw_Cdots:
4228     \group_end:
4229 }
4230 }

```

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`.

```

4231 \cs_new_protected:Npn \@@_actually_draw_Cdots:
4232 {
4233     \bool_if:NTF \l_@@_initial_open_bool
4234     { \@@_open_x_initial_dim: }
4235     { \@@_set_initial_coords_from_anchor:n { mid~east } }
4236     \bool_if:NTF \l_@@_final_open_bool
4237     { \@@_open_x_final_dim: }
4238     { \@@_set_final_coords_from_anchor:n { mid~west } }
4239     \bool_lazy_and:nnTF
4240     \l_@@_initial_open_bool
4241     \l_@@_final_open_bool
4242     {
4243         \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int }
4244         \dim_set_eq:NN \l_tmpa_dim \pgf@y
4245         \@@_qpoint:n { row - \int_eval:n { \l_@@_initial_i_int + 1 } }
4246         \dim_set:Nn \l_@@_y_initial_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
4247         \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
4248     }
4249     {
4250         \bool_if:NT \l_@@_initial_open_bool
4251         { \dim_set_eq:NN \l_@@_y_initial_dim \l_@@_y_final_dim }
4252         \bool_if:NT \l_@@_final_open_bool
4253         { \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim }
4254     }
4255     \@@_draw_line:
4256 }
4257 \cs_new_protected:Npn \@@_open_y_initial_dim:
4258 {
4259     \dim_set:Nn \l_@@_y_initial_dim { - \c_max_dim }
4260     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
4261     {
4262         \cs_if_exist:cT
4263         { pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4264         {
4265             \pgfpointanchor
4266             { \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4267             { north }
4268             \dim_set:Nn \l_@@_y_initial_dim

```



```

4269         { \dim_max:nn \l_@@_y_initial_dim \pgf@y }
4270     }
4271 }
4272 % modified 2023-08-10
4273 \dim_compare:nNnT \l_@@_y_initial_dim = { - \c_max_dim }
4274 {
4275     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4276     \dim_set:Nn \l_@@_y_initial_dim
4277     {
4278         \fp_to_dim:n
4279         {
4280             \pgf@y
4281             + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
4282         }
4283     }
4284 }
4285 }
4286 \cs_new_protected:Npn \@@_open_y_final_dim:
4287 {
4288     \dim_set_eq:NN \l_@@_y_final_dim \c_max_dim
4289     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
4290     {
4291         \cs_if_exist:cT
4292         { \pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4293         {
4294             \pgfpointanchor
4295             { \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4296             { south }
4297             \dim_set:Nn \l_@@_y_final_dim
4298             { \dim_min:nn \l_@@_y_final_dim \pgf@y }
4299         }
4300     }
4301 % modified 2023-08-10
4302 \dim_compare:nNnT \l_@@_y_final_dim = \c_max_dim
4303 {
4304     \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4305     \dim_set:Nn \l_@@_y_final_dim
4306     { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
4307 }
4308 }

```

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.

```

4309 \cs_new_protected:Npn \@@_draw_Vdots:nnn #1 #2 #3
4310 {
4311     \@@_adjust_to_submatrix:nn { #1 } { #2 }
4312     \cs_if_free:cT { @@ _dotted _ #1 - #2 }
4313     {
4314         \@@_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.

```

4315     \group_begin:
4316     \@@_open_shorten:
4317     \int_compare:nNnTF { #2 } = 0
4318     { \color { nicematrix-first-col } }
4319     {
4320         \int_compare:nNnT { #2 } = \l_@@_last_col_int
4321         { \color { nicematrix-last-col } }
4322     }
4323     \keys_set:nn { NiceMatrix / xdots } { #3 }
4324     \tl_if_empty:VF \l_@@_xdots_color_tl
4325     { \color { \l_@@_xdots_color_tl } }
4326     \@@_actually_draw_Vdots:

```

```

4327     \group_end:
4328   }
4329 }

```

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`.

```

4330 \cs_new_protected:Npn \@@_actually_draw_Vdots:
4331 {

```

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

```

4332   \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.

```

4333   {
4334     \@@_open_y_initial_dim:
4335     \@@_open_y_final_dim:
4336     \int_compare:nNnTF \l_@@_initial_j_int = \c_zero_int

```

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

```

4337     {
4338       \@@_qpoint:n { col - 1 }
4339       \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4340       \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
4341       \dim_sub:Nn \l_@@_x_initial_dim \l_@@_extra_left_margin_dim
4342       % \bool_if:NT \g_@@_delims_bool
4343       % {
4344         \dim_sub:Nn \l_@@_x_initial_dim \c_@@_shift_exterior_Vdots_dim
4345       % }
4346     }
4347     {
4348       \bool_lazy_and:nnTF
4349       { \int_compare_p:nNn \l_@@_last_col_int > { -2 } }
4350       { \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”.

```

4351     {
4352       \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4353       \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4354       \dim_add:Nn \l_@@_x_initial_dim \l_@@_right_margin_dim
4355       \dim_add:Nn \l_@@_x_initial_dim \l_@@_extra_right_margin_dim
4356       % \bool_if:NT \g_@@_delims_bool
4357       % {
4358         \dim_add:Nn
4359         \l_@@_x_initial_dim
4360         \c_@@_shift_exterior_Vdots_dim
4361       % }
4362     }

```

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

```

4363     {
4364       \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4365       \dim_set_eq:NN \l_tmpa_dim \pgf@x
4366       \@@_qpoint:n { col - \int_eval:n { \l_@@_initial_j_int + 1 } }

```

```

4367         \dim_set:Nn \l_@@_x_initial_dim { ( \pgf@x + \l_tmpa_dim ) / 2 }
4368     }
4369 }
4370 }

```

Now, the dotted line is *not* open on both sides (maybe open on only one side).

The boolean `\l_tmpa_bool` will indicate whether the column is of type 1 or may be considered as if.

```

4371 {
4372     \bool_set_false:N \l_tmpa_bool
4373     \bool_lazy_and:nnT
4374     { ! \l_@@_initial_open_bool }
4375     { ! \l_@@_final_open_bool }
4376     {
4377         \@@_set_initial_coords_from_anchor:n { south-west }
4378         \@@_set_final_coords_from_anchor:n { north-west }
4379         \bool_set:Nn \l_tmpa_bool
4380         { \dim_compare_p:nNn \l_@@_x_initial_dim = \l_@@_x_final_dim }
4381     }

```

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

```

4382     \bool_if:NTF \l_@@_initial_open_bool
4383     {
4384         \@@_open_y_initial_dim:
4385         \@@_set_final_coords_from_anchor:n { north }
4386         \dim_set_eq:NN \l_@@_x_initial_dim \l_@@_x_final_dim
4387     }
4388     {
4389         \@@_set_initial_coords_from_anchor:n { south }
4390         \bool_if:NTF \l_@@_final_open_bool
4391         \@@_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.

```

4392     {
4393         \@@_set_final_coords_from_anchor:n { north }
4394         \dim_compare:nNnF \l_@@_x_initial_dim = \l_@@_x_final_dim
4395         {
4396             \dim_set:Nn \l_@@_x_initial_dim
4397             {
4398                 \bool_if:NTF \l_tmpa_bool \dim_min:nn \dim_max:nn
4399                 \l_@@_x_initial_dim \l_@@_x_final_dim
4400             }
4401         }
4402     }
4403 }
4404 }
4405 \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
4406 \@@_draw_line:
4407 }

```

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.

```

4408 \cs_new_protected:Npn \@@_draw_Ddots:nnn #1 #2 #3
4409 {
4410     \@@_adjust_to_submatrix:nn { #1 } { #2 }
4411     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4412     {
4413         \@@_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.

```

4414     \group_begin:
4415     \@@_open_shorten:
4416     \keys_set:nn { NiceMatrix / xdots } { #3 }
4417     \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4418     \@@_actually_draw_Ddots:
4419     \group_end:
4420   }
4421 }

```

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.`

```

4422 \cs_new_protected:Npn \@@_actually_draw_Ddots:
4423 {
4424   \bool_if:NTF \l_@@_initial_open_bool
4425   {
4426     \@@_open_y_initial_dim:
4427     \@@_open_x_initial_dim:
4428   }
4429   { \@@_set_initial_coords_from_anchor:n { south-east } }
4430   \bool_if:NTF \l_@@_final_open_bool
4431   {
4432     \@@_open_x_final_dim:
4433     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4434   }
4435   { \@@_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.

```

4436   \bool_if:NT \l_@@_parallelize_diags_bool
4437   {
4438     \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).

```

4439     \int_compare:nNnTF \g_@@_ddots_int = 1

```

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.

```

4440     {
4441       \dim_gset:Nn \g_@@_delta_x_one_dim
4442       { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4443       \dim_gset:Nn \g_@@_delta_y_one_dim
4444       { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4445     }

```

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`.

```

4446     {
4447       \dim_set:Nn \l_@@_y_final_dim
4448       {
4449         \l_@@_y_initial_dim +

```

```

4450             ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4451             \dim_ratio:nn \g_@@_delta_y_one_dim \g_@@_delta_x_one_dim
4452         }
4453     }
4454 }
4455 \@@_draw_line:
4456 }

```

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.

```

4457 \cs_new_protected:Npn \@@_draw_Iddots:nnn #1 #2 #3
4458 {
4459     \@@_adjust_to_submatrix:nn { #1 } { #2 }
4460     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4461     {
4462         \@@_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.

```

4463     \group_begin:
4464     \@@_open_shorten:
4465     \keys_set:nn { NiceMatrix / xdots } { #3 }
4466     \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4467     \@@_actually_draw_Iddots:
4468     \group_end:
4469 }
4470 }

```

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`.

```

4471 \cs_new_protected:Npn \@@_actually_draw_Iddots:
4472 {
4473     \bool_if:NTF \l_@@_initial_open_bool
4474     {
4475         \@@_open_y_initial_dim:
4476         \@@_open_x_initial_dim:
4477     }
4478     { \@@_set_initial_coords_from_anchor:n { south-west } }
4479     \bool_if:NTF \l_@@_final_open_bool
4480     {
4481         \@@_open_y_final_dim:
4482         \@@_open_x_final_dim:
4483     }
4484     { \@@_set_final_coords_from_anchor:n { north-east } }
4485     \bool_if:NT \l_@@_parallelize_diags_bool
4486     {
4487         \int_gincr:N \g_@@_iddots_int
4488         \int_compare:nNnTF \g_@@_iddots_int = 1
4489         {
4490             \dim_gset:Nn \g_@@_delta_x_two_dim
4491             { \l_@@_x_final_dim - \l_@@_x_initial_dim }

```

```

4492         \dim_gset:Nn \g_@@_delta_y_two_dim
4493         { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4494     }
4495     {
4496         \dim_set:Nn \l_@@_y_final_dim
4497         {
4498             \l_@@_y_initial_dim +
4499             ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4500             \dim_ratio:nn \g_@@_delta_y_two_dim \g_@@_delta_x_two_dim
4501         }
4502     }
4503 }
4504 \@@_draw_line:
4505 }

```

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`

```

4506 \cs_new_protected:Npn \@@_draw_line:
4507 {
4508     \pgfrememberpicturepositiononpagetrue
4509     \pgf@relevantforpicturesizefalse
4510     \bool_lazy_or:nnTF
4511     { \tl_if_eq_p:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl }
4512     \l_@@_dotted_bool
4513     \@@_draw_standard_dotted_line:
4514     \@@_draw_unstandard_dotted_line:
4515 }

```

We have to do a special construction with `\exp_args:NV` to be able to put in the list of options in the correct place in the Tikz instruction.

```

4516 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:
4517 {
4518     \begin { scope }
4519     \@@_draw_unstandard_dotted_line:o
4520     { \l_@@_xdots_line_style_tl , \l_@@_xdots_color_tl }
4521 }

```

We have used the fact that, in PGF, un color name can be put directly in a list of options (that's why we have put directly `\l_@@_xdots_color_tl`).

The argument of `\@@_draw_unstandard_dotted_line:n` is, in fact, the list of options.

```

4522 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:n #1
4523 {
4524     \@@_draw_unstandard_dotted_line:nVWV
4525     { #1 }

```

```

4526     \l_@@_xdots_up_tl
4527     \l_@@_xdots_down_tl
4528     \l_@@_xdots_middle_tl
4529   }
4530 \cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:n { o }

```

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

```

4531 \hook_gput_code:nnn { begindocument } { . }
4532 {
4533   \IfPackageLoadedTF { tikz }
4534   {
4535     \tikzset
4536     {
4537       @@_node_above / .style = { sloped , above } ,
4538       @@_node_below / .style = { sloped , below } ,
4539       @@_node_middle / .style =
4540       {
4541         sloped ,
4542         inner~sep = \c_@@_innersep_middle_dim
4543       }
4544     }
4545   }
4546   { }
4547 }

4548 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:nmmm #1 #2 #3 #4
4549 {

```

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.

```

4550   \dim_zero_new:N \l_@@_l_dim
4551   \dim_set:Nn \l_@@_l_dim
4552   {
4553     \fp_to_dim:n
4554     {
4555       sqrt
4556       (
4557         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
4558         +
4559         ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
4560       )
4561     }
4562   }
4563   \bool_lazy_and:nnT % security
4564   { \dim_compare_p:nNn { \dim_abs:n \l_@@_l_dim } < \c_@@_max_l_dim }
4565   { \dim_compare_p:nNn { \dim_abs:n \l_@@_l_dim } > { 1 pt } }
4566   {
4567     \dim_set:Nn \l_tmpa_dim
4568     {
4569       \l_@@_x_initial_dim
4570       + ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
4571       * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
4572     }
4573     \dim_set:Nn \l_tmpb_dim
4574     {
4575       \l_@@_y_initial_dim
4576       + ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
4577       * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim

```

```

4578     }
4579     \dim_set:Nn \l_@@_tmpc_dim
4580     {
4581         \l_@@_x_final_dim
4582         - ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
4583         * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
4584     }
4585     \dim_set:Nn \l_@@_tmpd_dim
4586     {
4587         \l_@@_y_final_dim
4588         - ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
4589         * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
4590     }
4591     \dim_set_eq:NN \l_@@_x_initial_dim \l_tmpa_dim
4592     \dim_set_eq:NN \l_@@_y_initial_dim \l_tmpb_dim
4593     \dim_set_eq:NN \l_@@_x_final_dim \l_@@_tmpc_dim
4594     \dim_set_eq:NN \l_@@_y_final_dim \l_@@_tmpd_dim
4595 }

```

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

```

4596     \bool_if:NT \l_@@_xdots_h_labels_bool
4597     {
4598         \tikzset
4599         {
4600             @@_node_above / .style = { auto = left } ,
4601             @@_node_below / .style = { auto = right } ,
4602             @@_node_middle / .style = { inner~sep = \c_@@_innersep_middle_dim }
4603         }
4604     }
4605     \tl_if_empty:nF { #4 }
4606     { \tikzset { @@_node_middle / .append-style = { fill = white } } }
4607     \draw
4608     [ #1 ]
4609     ( \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).

```

4610     -- node [ @@_node_middle ] { $ \scriptstyle #4 $ }
4611     node [ @@_node_below ] { $ \scriptstyle #3 $ }
4612     node [ @@_node_above ] { $ \scriptstyle #2 $ }
4613     ( \l_@@_x_final_dim , \l_@@_y_final_dim ) ;
4614     \end { scope }
4615 }
4616 \cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:nmmn { n V V V }

```

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

```

4617 \cs_new_protected:Npn \@@_draw_standard_dotted_line:
4618 {
4619     \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.

```

4620     \dim_zero_new:N \l_@@_l_dim
4621     \dim_set:Nn \l_@@_l_dim
4622     {
4623         \fp_to_dim:n
4624         {
4625             sqrt
4626             (
4627                 ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
4628                 +
4629                 ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
4630             )

```



```

4631     }
4632 }

```

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.

```

4633     \bool_lazy_or:nmF
4634     { \dim_compare_p:nNn { \dim_abs:n \l_@@_l_dim } > \c_@@_max_l_dim }
4635     { \dim_compare_p:nNn \l_@@_l_dim = \c_zero_dim }
4636     \@@_draw_standard_dotted_line_i:
4637 \group_end:
4638 \bool_lazy_all:nF
4639 {
4640   { \tl_if_empty_p:N \l_@@_xdots_up_tl }
4641   { \tl_if_empty_p:N \l_@@_xdots_down_tl }
4642   { \tl_if_empty_p:N \l_@@_xdots_middle_tl }
4643 }
4644 \l_@@_labels_standard_dotted_line:
4645 }
4646 \dim_const:Nn \c_@@_max_l_dim { 50 cm }
4647 \cs_new_protected:Npn \@@_draw_standard_dotted_line_i:
4648 {

```

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

```

4649   \int_set:Nn \l_tmpa_int
4650   {
4651     \dim_ratio:nn
4652     {
4653       \l_@@_l_dim
4654       - \l_@@_xdots_shorten_start_dim
4655       - \l_@@_xdots_shorten_end_dim
4656     }
4657     \l_@@_xdots_inter_dim
4658   }

```

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

```

4659   \dim_set:Nn \l_tmpa_dim
4660   {
4661     ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4662     \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
4663   }
4664   \dim_set:Nn \l_tmpb_dim
4665   {
4666     ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
4667     \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
4668   }

```

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.

```

4669   \dim_gadd:Nn \l_@@_x_initial_dim
4670   {
4671     ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4672     \dim_ratio:nn
4673     {
4674       \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
4675       + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
4676     }
4677     { 2 \l_@@_l_dim }
4678   }
4679   \dim_gadd:Nn \l_@@_y_initial_dim

```

```

4680     {
4681     ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
4682     \dim_ratio:nn
4683     {
4684     \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
4685     + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
4686     }
4687     { 2 \l_@@_l_dim }
4688     }
4689     \pgf@relevantforpicturesizefalse
4690     \int_step_inline:nnn 0 \l_tmpa_int
4691     {
4692     \pgfpathcircle
4693     { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
4694     { \l_@@_xdots_radius_dim }
4695     \dim_add:Nn \l_@@_x_initial_dim \l_tmpa_dim
4696     \dim_add:Nn \l_@@_y_initial_dim \l_tmpb_dim
4697     }
4698     \pgfusepathqfill
4699     }

```

```

4700 \cs_new_protected:Npn \l_@@_labels_standard_dotted_line:
4701 {
4702     \pgfscope
4703     \pgftransformshift
4704     {
4705     \pgfpointlineattime { 0.5 }
4706     { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
4707     { \pgfpoint \l_@@_x_final_dim \l_@@_y_final_dim }
4708     }
4709     \fp_set:Nn \l_tmpa_fp
4710     {
4711     atand
4712     (
4713     \l_@@_y_final_dim - \l_@@_y_initial_dim ,
4714     \l_@@_x_final_dim - \l_@@_x_initial_dim
4715     )
4716     }
4717     \pgftransformrotate { \fp_use:N \l_tmpa_fp }
4718     \bool_if:NF \l_@@_xdots_h_labels_bool { \fp_zero:N \l_tmpa_fp }
4719     \tl_if_empty:NF \l_@@_xdots_middle_tl
4720     {
4721     \begin { pgfscope }
4722     \pgfset { inner-sep = \c_@@_innersep_middle_dim }
4723     \pgfnode
4724     { rectangle }
4725     { center }
4726     {
4727     \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4728     {
4729     \c_math_toggle_token
4730     \scriptstyle \l_@@_xdots_middle_tl
4731     \c_math_toggle_token
4732     }
4733     }
4734     { }
4735     {
4736     \pgfsetfillcolor { white }
4737     \pgfusepath { fill }
4738     }
4739     \end { pgfscope }
4740     }
4741     \tl_if_empty:NF \l_@@_xdots_up_tl

```

```

4742     {
4743     \pgfnode
4744     { rectangle }
4745     { south }
4746     {
4747     \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4748     {
4749     \c_math_toggle_token
4750     \scriptstyle \l_@@_xdots_up_tl
4751     \c_math_toggle_token
4752     }
4753     }
4754     { }
4755     { \pgfusepath { } }
4756     }
4757 \tl_if_empty:NF \l_@@_xdots_down_tl
4758 {
4759 \pgfnode
4760 { rectangle }
4761 { north }
4762 {
4763 \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4764 {
4765 \c_math_toggle_token
4766 \scriptstyle \l_@@_xdots_down_tl
4767 \c_math_toggle_token
4768 }
4769 }
4770 { }
4771 { \pgfusepath { } }
4772 }
4773 \endpgfscope
4774 }

```

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.

```

4775 \hook_gput_code:nnn { begindocument } { . }
4776 {
4777 \tl_set:Nn \l_@@_argspec_tl { m E { _ ^ : } { { } { } { } } }
4778 \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
4779 \cs_new_protected:Npn \@@_Ldots
4780 { \@@_collect_options:n { \@@_Ldots_i } }
4781 \exp_args:NNV \NewDocumentCommand \@@_Ldots_i \l_@@_argspec_tl
4782 {
4783 \int_compare:nNnTF \c@jCol = 0
4784 { \@@_error:nn { in-first-col } \Ldots }
4785 {
4786 \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
4787 { \@@_error:nn { in-last-col } \Ldots }
4788 {

```

```

4789         \@@_instruction_of_type:nnn \c_false_bool { Ldots }
4790         { #1 , down = #2 , up = #3 , middle = #4 }
4791     }
4792 }
4793 \bool_if:NF \l_@@_nullify_dots_bool
4794 { \phantom { \ensuremath { \@@_old_ldots } } }
4795 \bool_gset_true:N \g_@@_empty_cell_bool
4796 }

4797 \cs_new_protected:Npn \@@_Cdots
4798 { \@@_collect_options:n { \@@_Cdots_i } }
4799 \exp_args:NNV \NewDocumentCommand \@@_Cdots_i \l_@@_argspec_tl
4800 {
4801     \int_compare:nNnTF \c@jCol = 0
4802     { \@@_error:nn { in~first~col } \Cdots }
4803     {
4804         \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
4805         { \@@_error:nn { in~last~col } \Cdots }
4806         {
4807             \@@_instruction_of_type:nnn \c_false_bool { Cdots }
4808             { #1 , down = #2 , up = #3 , middle = #4 }
4809         }
4810     }
4811     \bool_if:NF \l_@@_nullify_dots_bool
4812     { \phantom { \ensuremath { \@@_old_cdots } } }
4813     \bool_gset_true:N \g_@@_empty_cell_bool
4814 }

4815 \cs_new_protected:Npn \@@_Vdots
4816 { \@@_collect_options:n { \@@_Vdots_i } }
4817 \exp_args:NNV \NewDocumentCommand \@@_Vdots_i \l_@@_argspec_tl
4818 {
4819     \int_compare:nNnTF \c@iRow = 0
4820     { \@@_error:nn { in~first~row } \Vdots }
4821     {
4822         \int_compare:nNnTF \c@iRow = \l_@@_last_row_int
4823         { \@@_error:nn { in~last~row } \Vdots }
4824         {
4825             \@@_instruction_of_type:nnn \c_false_bool { Vdots }
4826             { #1 , down = #2 , up = #3 , middle = #4 }
4827         }
4828     }
4829     \bool_if:NF \l_@@_nullify_dots_bool
4830     { \phantom { \ensuremath { \@@_old_vdots } } }
4831     \bool_gset_true:N \g_@@_empty_cell_bool
4832 }

4833 \cs_new_protected:Npn \@@_Ddots
4834 { \@@_collect_options:n { \@@_Ddots_i } }
4835 \exp_args:NNV \NewDocumentCommand \@@_Ddots_i \l_@@_argspec_tl
4836 {
4837     \int_case:nnF \c@iRow
4838     {
4839         0 { \@@_error:nn { in~first~row } \Ddots }
4840         \l_@@_last_row_int { \@@_error:nn { in~last~row } \Ddots }
4841     }
4842     {
4843         \int_case:nnF \c@jCol
4844         {
4845             0 { \@@_error:nn { in~first~col } \Ddots }
4846             \l_@@_last_col_int { \@@_error:nn { in~last~col } \Ddots }

```

```

4847     }
4848     {
4849         \keys_set_known:nn { NiceMatrix / Ddots } { #1 }
4850         \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Ddots }
4851         { #1 , down = #2 , up = #3 , middle = #4 }
4852     }
4853
4854 }
4855 \bool_if:NF \l_@@_nullify_dots_bool
4856 { \phantom { \ensuremath { \@@_old_ddots } } }
4857 \bool_gset_true:N \g_@@_empty_cell_bool
4858 }

4859 \cs_new_protected:Npn \@@_Iddots
4860 { \@@_collect_options:n { \@@_Iddots_i } }
4861 \exp_args:NNV \NewDocumentCommand \@@_Iddots_i \l_@@_argspec_tl
4862 {
4863     \int_case:nnF \c@iRow
4864     {
4865         0 { \@@_error:nn { in~first~row } \Iddots }
4866         \l_@@_last_row_int { \@@_error:nn { in~last~row } \Iddots }
4867     }
4868     {
4869         \int_case:nnF \c@jCol
4870         {
4871             0 { \@@_error:nn { in~first~col } \Iddots }
4872             \l_@@_last_col_int { \@@_error:nn { in~last~col } \Iddots }
4873         }
4874         {
4875             \keys_set_known:nn { NiceMatrix / Ddots } { #1 }
4876             \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Iddots }
4877             { #1 , down = #2 , up = #3 , middle = #4 }
4878         }
4879     }
4880     \bool_if:NF \l_@@_nullify_dots_bool
4881     { \phantom { \ensuremath { \@@_old_iddots } } }
4882     \bool_gset_true:N \g_@@_empty_cell_bool
4883 }
4884 }

```

End of the `\AddToHook`.

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

```

4885 \keys_define:nn { NiceMatrix / Ddots }
4886 {
4887     draw-first .bool_set:N = \l_@@_draw_first_bool ,
4888     draw-first .default:n = true ,
4889     draw-first .value_forbidden:n = true
4890 }

```

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

```

4891 \cs_new_protected:Npn \@@_Hspace:
4892 {
4893     \bool_gset_true:N \g_@@_empty_cell_bool
4894     \hspace
4895 }

```

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`.

```

4896 \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`.

```

4897 \cs_new:Npn \@@_Hdotsfor:
4898 {
4899   \bool_lazy_and:nnTF
4900     { \int_compare_p:nNn \c@jCol = 0 }
4901     { \int_compare_p:nNn \l_@@_first_col_int = 0 }
4902     {
4903       \bool_if:NTF \g_@@_after_col_zero_bool
4904         {
4905           \multicolumn { 1 } { c } { }
4906           \@@_Hdotsfor_i
4907         }
4908       { \@@_fatal:n { Hdotsfor~in~col~0 } }
4909     }
4910   {
4911     \multicolumn { 1 } { c } { }
4912     \@@_Hdotsfor_i
4913   }
4914 }

```

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:`).

```

4915 \hook_gput_code:nnn { begindocument } { . }
4916 {
4917   \tl_set:Nn \l_@@_argspec_tl { m m 0 { } E { _ ^ : } { { } { } { } } }
4918   \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.

```

4919   \cs_new_protected:Npn \@@_Hdotsfor_i
4920     { \@@_collect_options:n { \@@_Hdotsfor_ii } }
4921   \exp_args:NNV \NewDocumentCommand \@@_Hdotsfor_ii \l_@@_argspec_tl
4922     {
4923       \tl_gput_right:Nx \g_@@_HVdotsfor_lines_tl
4924         {
4925           \@@_Hdotsfor:nnnn
4926             { \int_use:N \c@iRow }
4927             { \int_use:N \c@jCol }
4928             { #2 }
4929             {
4930               #1 , #3 ,
4931               down = \exp_not:n { #4 } ,
4932               up = \exp_not:n { #5 } ,
4933               middle = \exp_not:n { #6 }
4934             }
4935           }
4936       \prg_replicate:nn { #2 - 1 } { & \multicolumn { 1 } { c } { } }
4937     }
4938 }

```

```

4939 \cs_new_protected:Npn \@@_Hdotsfor:nnnn #1 #2 #3 #4
4940 {
4941   \bool_set_false:N \l_@@_initial_open_bool
4942   \bool_set_false:N \l_@@_final_open_bool

```

For the row, it's easy.

```

4943   \int_set:Nn \l_@@_initial_i_int { #1 }
4944   \int_set_eq:NN \l_@@_final_i_int \l_@@_initial_i_int

```

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

```

4945 \int_compare:nNnTF { #2 } = 1
4946 {
4947   \int_set:Nn \l_@@_initial_j_int 1
4948   \bool_set_true:N \l_@@_initial_open_bool
4949 }
4950 {
4951   \cs_if_exist:cTF
4952   {
4953     pgf @ sh @ ns @ \@@_env:
4954     - \int_use:N \l_@@_initial_i_int
4955     - \int_eval:n { #2 - 1 }
4956   }
4957   { \int_set:Nn \l_@@_initial_j_int { #2 - 1 } }
4958   {
4959     \int_set:Nn \l_@@_initial_j_int { #2 }
4960     \bool_set_true:N \l_@@_initial_open_bool
4961   }
4962 }
4963 \int_compare:nNnTF { #2 + #3 - 1 } = \c@jCol
4964 {
4965   \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
4966   \bool_set_true:N \l_@@_final_open_bool
4967 }
4968 {
4969   \cs_if_exist:cTF
4970   {
4971     pgf @ sh @ ns @ \@@_env:
4972     - \int_use:N \l_@@_final_i_int
4973     - \int_eval:n { #2 + #3 }
4974   }
4975   { \int_set:Nn \l_@@_final_j_int { #2 + #3 } }
4976   {
4977     \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
4978     \bool_set_true:N \l_@@_final_open_bool
4979   }
4980 }
4981 \group_begin:
4982
4983 \@@_open_shorten:
4984
4985
4986 \int_compare:nNnTF { #1 } = 0
4987 { \color { nicematrix-first-row } }
4988 {
4989   \int_compare:nNnT { #1 } = \g_@@_row_total_int
4990   { \color { nicematrix-last-row } }
4991 }
4992
4993 \keys_set:nn { NiceMatrix / xdots } { #4 }
4994 \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4995 \@@_actually_draw_Ldots:
4996 \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).

```

4997 \int_step_inline:nnn { #2 } { #2 + #3 - 1 }
4998 { \cs_set:cpn { @@ _ dotted _ #1 - ##1 } { } }
4999 }

```

```

5000 \hook_gput_code:nnn { begindocument } { . }

```

```

5001 {
5002   \tl_set:Nn \l_@@_argspec_tl { m m 0 { } E { _ ^ : } { { } { } { } } }
5003   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
5004   \cs_new_protected:Npn \@@_Vdotsfor:
5005     { \@@_collect_options:n { \@@_Vdotsfor_i } }
5006   \exp_args:NNV \NewDocumentCommand \@@_Vdotsfor_i \l_@@_argspec_tl
5007     {
5008     \bool_gset_true:N \g_@@_empty_cell_bool
5009     \tl_gput_right:Nx \g_@@_HVdotsfor_lines_tl
5010       {
5011         \@@_Vdotsfor:nmmn
5012         { \int_use:N \c@iRow }
5013         { \int_use:N \c@jCol }
5014         { #2 }
5015         {
5016           #1 , #3 ,
5017           down = \exp_not:n { #4 } ,
5018           up = \exp_not:n { #5 } ,
5019           middle = \exp_not:n { #6 }
5020         }
5021       }
5022     }
5023 }

```

```

5024 \cs_new_protected:Npn \@@_Vdotsfor:nmmn #1 #2 #3 #4
5025 {
5026   \bool_set_false:N \l_@@_initial_open_bool
5027   \bool_set_false:N \l_@@_final_open_bool

```

For the column, it's easy.

```

5028   \int_set:Nn \l_@@_initial_j_int { #2 }
5029   \int_set_eq:NN \l_@@_final_j_int \l_@@_initial_j_int

```

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

```

5030   \int_compare:nNnTF { #1 } = 1
5031     {
5032       \int_set:Nn \l_@@_initial_i_int 1
5033       \bool_set_true:N \l_@@_initial_open_bool
5034     }
5035     {
5036       \cs_if_exist:cTF
5037         {
5038           pgf @ sh @ ns @ \@@_env:
5039           - \int_eval:n { #1 - 1 }
5040           - \int_use:N \l_@@_initial_j_int
5041         }
5042         { \int_set:Nn \l_@@_initial_i_int { #1 - 1 } }
5043         {
5044           \int_set:Nn \l_@@_initial_i_int { #1 }
5045           \bool_set_true:N \l_@@_initial_open_bool
5046         }
5047     }
5048   \int_compare:nNnTF { #1 + #3 - 1 } = \c@iRow
5049     {
5050       \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5051       \bool_set_true:N \l_@@_final_open_bool
5052     }
5053     {
5054       \cs_if_exist:cTF
5055         {
5056           pgf @ sh @ ns @ \@@_env:
5057           - \int_eval:n { #1 + #3 }
5058           - \int_use:N \l_@@_final_j_int

```



```

5059     }
5060     { \int_set:Nn \l_@@_final_i_int { #1 + #3 } }
5061     {
5062         \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5063         \bool_set_true:N \l_@@_final_open_bool
5064     }
5065 }
5066 \group_begin:
5067 \@@_open_shorten:
5068
5069
5070
5071
5072 \int_compare:nNnTF { #2 } = 0
5073 { \color { nicematrix-first-col } }
5074 {
5075     \int_compare:nNnT { #2 } = \g_@@_col_total_int
5076     { \color { nicematrix-last-col } }
5077 }
5078 \keys_set:nn { NiceMatrix / xdots } { #4 }
5079 \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
5080 \@@_actually_draw_Vdots:
5081 \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).

```

5082     \int_step_inline:nnn { #1 } { #1 + #3 - 1 }
5083     { \cs_set:cpn { @@ _ dotted _ ##1 - #2 } { } }
5084 }

```

The command `\@@_rotate:` will be linked to `\rotate` in `{NiceArrayWithDelims}`.

```

5085 \NewDocumentCommand \@@_rotate: { 0 { } }
5086 {
5087     \peek_remove_spaces:n
5088     {
5089         \bool_gset_true:N \g_@@_rotate_bool
5090         \keys_set:nn { NiceMatrix / rotate } { #1 }
5091     }
5092 }
5093 \keys_define:nn { NiceMatrix / rotate }
5094 {
5095     c .code:n = \bool_gset_true:N \g_@@_rotate_c_bool ,
5096     c .value_forbidden:n = true ,
5097     unknown .code:n = \@@_error:n { Unknown-key-for-rotate }
5098 }

```

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

```

5099 \cs_new:Npn \@_double_int_eval:n #1-#2 \q_stop
5100   {
5101     \tl_if_empty:nTF { #2 }
5102       { #1 }
5103       { \@_double_int_eval_i:n #1-#2 \q_stop }
5104   }
5105 \cs_new:Npn \@_double_int_eval_i:n #1-#2- \q_stop
5106   { \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).

```

5107 \hook_gput_code:nnn { begindocument } { . }
5108   {
5109     \tl_set:Nn \l_@@_argspec_tl { 0 { } m m ! 0 { } E { _ ^ : } { { } { } { } } }
5110     \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
5111     \exp_args:NNV \NewDocumentCommand \@_line \l_@@_argspec_tl
5112       {
5113         \group_begin:
5114         \keys_set:nn { NiceMatrix / xdots } { #1 , #4 , down = #5 , up = #6 }
5115         \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
5116         \use:e
5117           {
5118             \@_line_i:nn
5119             { \@_double_int_eval:n #2 - \q_stop }
5120             { \@_double_int_eval:n #3 - \q_stop }
5121           }
5122         \group_end:
5123       }
5124   }

5125 \cs_new_protected:Npn \@_line_i:nn #1 #2
5126   {
5127     \bool_set_false:N \l_@@_initial_open_bool
5128     \bool_set_false:N \l_@@_final_open_bool
5129     \bool_if:nTF
5130       {
5131         \cs_if_free_p:c { pgf @ sh @ ns @ \@_env: - #1 }
5132         ||
5133         \cs_if_free_p:c { pgf @ sh @ ns @ \@_env: - #2 }
5134       }
5135       {
5136         \@_error:nnn { unknown~cell~for~line~in~CodeAfter } { #1 } { #2 }
5137       }

```

The test of `measuring@` is a security (cf. question 686649 on TeX StackExchange).

```

5138   { \legacy_if:nF { measuring@ } { \@_draw_line_ii:nn { #1 } { #2 } } }
5139   }

5140 \hook_gput_code:nnn { begindocument } { . }
5141   {
5142     \cs_new_protected:Npx \@_draw_line_ii:nn #1 #2
5143     {

```

¹³Indeed, we want that the user may use the command `\line` in `\CodeAfter` with LaTeX counters in the arguments — with the command `\value`.

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:`.

```

5144     \c_@@_pgfortikzpicture_tl
5145     \@@_draw_line_iii:nn { #1 } { #2 }
5146     \c_@@_endpgfortikzpicture_tl
5147   }
5148 }

```

The following command *must* be protected (it’s used in the construction of `\@@_draw_line_ii:nn`).

```

5149 \cs_new_protected:Npn \@@_draw_line_iii:nn #1 #2
5150 {
5151   \pgfrememberpicturepositiononpagetrue
5152   \pgfpointshapeborder { \@@_env: - #1 } { \@@_qpoint:n { #2 } }
5153   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
5154   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
5155   \pgfpointshapeborder { \@@_env: - #2 } { \@@_qpoint:n { #1 } }
5156   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
5157   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
5158   \@@_draw_line:
5159 }

```

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`

```

5160 \keys_define:nn { NiceMatrix / RowStyle }
5161 {
5162   cell-space-top-limit .dim_set:N = \l_tmpa_dim ,
5163   cell-space-top-limit .initial:n = \c_zero_dim ,
5164   cell-space-top-limit .value_required:n = true ,
5165   cell-space-bottom-limit .dim_set:N = \l_tmpb_dim ,
5166   cell-space-bottom-limit .initial:n = \c_zero_dim ,
5167   cell-space-bottom-limit .value_required:n = true ,
5168   cell-space-limits .meta:n =
5169   {
5170     cell-space-top-limit = #1 ,
5171     cell-space-bottom-limit = #1 ,
5172   } ,
5173   color .tl_set:N = \l_@@_color_tl ,
5174   color .value_required:n = true ,
5175   bold .bool_set:N = \l_tmpa_bool ,
5176   bold .default:n = true ,
5177   bold .initial:n = false ,
5178   nb-rows .code:n =
5179   \str_if_eq:nnTF { #1 } { * }
5180   { \int_set:Nn \l_@@_key_nb_rows_int { 500 } }
5181   { \int_set:Nn \l_@@_key_nb_rows_int { #1 } } ,
5182   nb-rows .value_required:n = true ,
5183   rowcolor .tl_set:N = \l_tmpa_tl ,
5184   rowcolor .value_required:n = true ,
5185   rowcolor .initial:n = ,
5186   unknown .code:n = \@@_error:n { Unknown-key-for-RowStyle }
5187 }

5188 \NewDocumentCommand \@@_RowStyle:n { 0 { } m }
5189 {

```

```

5190 \group_begin:
5191 \tl_clear:N \l_tmpa_tl % value of \rowcolor
5192 \tl_clear:N \l_@@_color_tl
5193 \int_set:Nn \l_@@_key_nb_rows_int 1
5194 \keys_set:nn { NiceMatrix / RowStyle } { #1 }

```

If the key rowcolor has been used.

```

5195 \tl_if_empty:NF \l_tmpa_tl
5196 {

```

First, the end of the current row (we remind that \RowStyle applies to the *end* of the current row).

```

5197 \tl_gput_right:Nx \g_@@_pre_code_before_tl
5198 {

```

The command \@@_exp_color_arg:NV is *fully expandable*.

```

5199 \@@_exp_color_arg:NV \@@_rectanglecolor \l_tmpa_tl
5200 { \int_use:N \c@iRow - \int_use:N \c@jCol }
5201 { \int_use:N \c@iRow - * }
5202 }

```

Then, the other rows (if there is several rows).

```

5203 \int_compare:nNnT \l_@@_key_nb_rows_int > 1
5204 {
5205 \tl_gput_right:Nx \g_@@_pre_code_before_tl
5206 {
5207 \@@_exp_color_arg:NV \@@_rowcolor \l_tmpa_tl
5208 {
5209 \int_eval:n { \c@iRow + 1 }
5210 - \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int - 1 }
5211 }
5212 }
5213 }
5214 }
5215 \tl_gput_right:Nn \g_@@_row_style_tl { \ifnum \c@iRow < }
5216 \tl_gput_right:Nx \g_@@_row_style_tl
5217 { \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int } }
5218 \tl_gput_right:Nn \g_@@_row_style_tl { #2 }

```

\l_tmpa_dim is the value of the key cell-space-top-limit of \RowStyle.

```

5219 \dim_compare:nNnT \l_tmpa_dim > \c_zero_dim
5220 {
5221 \tl_gput_right:Nx \g_@@_row_style_tl
5222 {
5223 \tl_gput_right:Nn \exp_not:N \g_@@_cell_after_hook_tl
5224 {
5225 \dim_set:Nn \l_@@_cell_space_top_limit_dim
5226 { \dim_use:N \l_tmpa_dim }
5227 }
5228 }
5229 }

```

\l_tmpb_dim is the value of the key cell-space-bottom-limit of \RowStyle.

```

5230 \dim_compare:nNnT \l_tmpb_dim > \c_zero_dim
5231 {
5232 \tl_gput_right:Nx \g_@@_row_style_tl
5233 {
5234 \tl_gput_right:Nn \exp_not:N \g_@@_cell_after_hook_tl
5235 {
5236 \dim_set:Nn \l_@@_cell_space_bottom_limit_dim
5237 { \dim_use:N \l_tmpb_dim }
5238 }
5239 }
5240 }

```

\l_@@_color_tl is the value of the key color of \RowStyle.

```

5241 \tl_if_empty:NF \l_@@_color_tl
5242 {
5243 \tl_gput_right:Nx \g_@@_row_style_tl

```

```

5244     {
5245     \mode_leave_vertical:
5246     \@@_color:n { \l_@@_color_tl }
5247     }
5248   }
\l_tmpa_bool is the value of the key bold.
5249   \bool_if:NT \l_tmpa_bool
5250   {
5251     \tl_gput_right:Nn \g_@@_row_style_tl
5252     {
5253       \if_mode_math:
5254         \c_math_toggle_token
5255         \bfseries \boldmath
5256         \c_math_toggle_token
5257       \else:
5258         \bfseries \boldmath
5259       \fi:
5260     }
5261   }
5262   \tl_gput_right:Nn \g_@@_row_style_tl { \fi }
5263   \group_end:
5264   \g_@@_row_style_tl
5265   \ignorespaces
5266 }

```

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).

```

5267 \cs_new_protected:Npn \@@_add_to_colors_seq:nn #1 #2
5268   {

```

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.

```

5269   \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`.

```

5270   \str_if_in:nnF { #1 } { !! }
5271   {
5272     \seq_map_indexed_inline:Nn \g_@@_colors_seq
5273     { \tl_if_eq:nnT { #1 } { ##2 } { \int_set:Nn \l_tmpa_int { ##1 } } }
5274   }
5275   \int_compare:nNnTF \l_tmpa_int = \c_zero_int

```

First, the case where the color is a *new* color (not in the sequence).

```

5276     {
5277         \seq_gput_right:Nn \g_@@_colors_seq { #1 }
5278         \tl_gset:cx { g_@@_color _ \seq_count:N \g_@@_colors_seq _ tl } { #2 }
5279     }

```

Now, the case where the color is *not* a new color (the color is in the sequence at the position `\l_tmpa_int`).

```

5280     { \tl_gput_right:cx { g_@@_color _ \int_use:N \l_tmpa_int _tl } { #2 } }
5281 }

5282 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { x n }
5283 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { x x }

```

The following command must be used within a `\pgfpicture`.

```

5284 \cs_new_protected:Npn \@@_clip_with_rounded_corners:
5285 {
5286     \dim_compare:nNnT \l_@@_tab_rounded_corners_dim > \c_zero_dim
5287     {

```

The TeX group is for `\pgfsetcornersarced` (whose scope is the TeX scope).

```

5288     \group_begin:
5289     \pgfsetcornersarced
5290     {
5291         \pgfpoint
5292         { \l_@@_tab_rounded_corners_dim }
5293         { \l_@@_tab_rounded_corners_dim }
5294     }

```

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.

```

5295     \bool_if:NTF \l_@@_hvlines_bool
5296     {
5297         \pgfpathrectanglecorners
5298         {
5299             \pgfpointadd
5300             { \@@_qpoint:n { row-1 } }
5301             { \pgfpoint { 0.5 \arrayrulewidth } { \c_zero_dim } }
5302         }
5303         {
5304             \pgfpointadd
5305             {
5306                 \@@_qpoint:n
5307                 { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
5308             }
5309             { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
5310         }
5311     }
5312     {
5313         \pgfpathrectanglecorners
5314         { \@@_qpoint:n { row-1 } }
5315         {
5316             \pgfpointadd
5317             {
5318                 \@@_qpoint:n
5319                 { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
5320             }
5321             { \pgfpoint \c_zero_dim \arrayrulewidth }
5322         }
5323     }
5324     \pgfusepath { clip }
5325     \group_end:

```

The TeX group was for `\pgfsetcornersarced`.

```
5326     }
5327 }
```

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`).

```
5328 \cs_new_protected:Npn \@@_actually_color:
5329 {
5330   \pgfpicture
5331   \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.

```
5332   \@@_clip_with_rounded_corners:
5333   \seq_map_indexed_inline:Nn \g_@@_colors_seq
5334   {
5335     \begin { pgfscope }
5336     \@@_color_opacity ##2
5337     \use:c { g_@@_color _ ##1 _tl }
5338     \tl_gclear:c { g_@@_color _ ##1 _tl }
5339     \pgfusepath { fill }
5340     \end { pgfscope }
5341   }
5342   \endpgfpicture
5343 }
```

The following command will extract the potential key `opacity` in its optional argument (between square brackets) and (of course) then apply the command `\color`.

```
5344 \cs_new_protected:Npn \@@_color_opacity
5345 {
5346   \peek_meaning:NTF [
5347     { \@@_color_opacity:w }
5348     { \@@_color_opacity:w [ ] }
5349   }
```

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.

```
5350 \cs_new_protected:Npn \@@_color_opacity:w [ #1 ]
5351 {
5352   \tl_clear:N \l_tmpa_tl
5353   \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.
5354   \tl_if_empty:NF \l_tmpa_tl { \exp_args:NV \pgfsetfillopacity \l_tmpa_tl }
5355   \tl_if_empty:NTF \l_tmpb_tl
5356     { \@declaredcolor }
5357     { \use:x { \exp_not:N \@undeclaredcolor [ \l_tmpb_tl ] } }
5358 }
```

The following set of keys is used by the command `\@@_color_opacity:wn`.

```
5359 \keys_define:nn { nicematrix / color-opacity }
5360 {
5361   opacity .tl_set:N          = \l_tmpa_tl ,
5362   opacity .value_required:n = true
5363 }
```

```

5364 \cs_new_protected:Npn \@@_cartesian_color:nn #1 #2
5365 {
5366   \tl_set:Nn \l_@@_rows_tl { #1 }
5367   \tl_set:Nn \l_@@_cols_tl { #2 }
5368   \@@_cartesian_path:
5369 }

```

Here is an example : `\@@_rowcolor {red!15} {1,3,5-7,10-}`

```

5370 \NewDocumentCommand \@@_rowcolor { 0 { } m m }
5371 {
5372   \tl_if_blank:nF { #2 }
5373   {
5374     \@@_add_to_colors_seq:xn
5375     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5376     { \@@_cartesian_color:nn { #3 } { - } }
5377   }
5378 }

```

Here an example : `\@@_columncolor:nn {red!15} {1,3,5-7,10-}`

```

5379 \NewDocumentCommand \@@_columncolor { 0 { } m m }
5380 {
5381   \tl_if_blank:nF { #2 }
5382   {
5383     \@@_add_to_colors_seq:xn
5384     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5385     { \@@_cartesian_color:nn { - } { #3 } }
5386   }
5387 }

```

Here is an example : `\@@_rectanglecolor{red!15}{2-3}{5-6}`

```

5388 \NewDocumentCommand \@@_rectanglecolor { 0 { } m m m }
5389 {
5390   \tl_if_blank:nF { #2 }
5391   {
5392     \@@_add_to_colors_seq:xn
5393     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5394     { \@@_rectanglecolor:nnn { #3 } { #4 } { 0 pt } }
5395   }
5396 }

```

The last argument is the radius of the corners of the rectangle.

```

5397 \NewDocumentCommand \@@_roundedrectanglecolor { 0 { } m m m m }
5398 {
5399   \tl_if_blank:nF { #2 }
5400   {
5401     \@@_add_to_colors_seq:xn
5402     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5403     { \@@_rectanglecolor:nnn { #3 } { #4 } { #5 } }
5404   }
5405 }

```

The last argument is the radius of the corners of the rectangle.

```

5406 \cs_new_protected:Npn \@@_rectanglecolor:nnn #1 #2 #3
5407 {
5408   \@@_cut_on_hyphen:w #1 \q_stop
5409   \tl_clear_new:N \l_@@_tmpc_tl
5410   \tl_clear_new:N \l_@@_tmpd_tl
5411   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
5412   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
5413   \@@_cut_on_hyphen:w #2 \q_stop
5414   \tl_set:Nx \l_@@_rows_tl { \l_@@_tmpc_tl - \l_tmpa_tl }
5415   \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`.

```
5416 \@@_cartesian_path:n { #3 }
5417 }
```

Here is an example: `\@@_cellcolor[rgb]{0.5,0.5,0}{2-3,3-4,4-5,5-6}`

```
5418 \NewDocumentCommand \@@_cellcolor { 0 { } m m }
5419 {
5420 \clist_map_inline:nn { #3 }
5421 { \@@_rectanglecolor [ #1 ] { #2 } { ##1 } { ##1 } }
5422 }
```

```
5423 \NewDocumentCommand \@@_chessboardcolors { 0 { } m m }
5424 {
5425 \int_step_inline:nn { \int_use:N \c@iRow }
5426 {
5427 \int_step_inline:nn { \int_use:N \c@jCol }
5428 {
5429 \int_if_even:nTF { ####1 + ##1 }
5430 { \@@_cellcolor [ #1 ] { #2 } }
5431 { \@@_cellcolor [ #1 ] { #3 } }
5432 { ##1 - ####1 }
5433 }
5434 }
5435 }
```

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”.

```
5436 \NewDocumentCommand \@@_arraycolor { 0 { } m }
5437 {
5438 \@@_rectanglecolor [ #1 ] { #2 }
5439 { 1 - 1 }
5440 { \int_use:N \c@iRow - \int_use:N \c@jCol }
5441 }
```

```
5442 \keys_define:nn { NiceMatrix / rowcolors }
5443 {
5444 respect-blocks .bool_set:N = \l_@@_respect_blocks_bool ,
5445 respect-blocks .default:n = true ,
5446 cols .tl_set:N = \l_@@_cols_tl ,
5447 restart .bool_set:N = \l_@@_rowcolors_restart_bool ,
5448 restart .default:n = true ,
5449 unknown .code:n = \@@_error:n { Unknown-key-for-rowcolors }
5450 }
```

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` apperas 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*.

```
5451 \NewDocumentCommand \@@_rowlistcolors { 0 { } m m 0 { } }
5452 {
```

The group is for the options. `\l_@@_colors_seq` will be the list of colors.

```

5453 \group_begin:
5454 \seq_clear_new:N \l_@@_colors_seq
5455 \seq_set_split:Nnn \l_@@_colors_seq { , } { #3 }
5456 \tl_clear_new:N \l_@@_cols_tl
5457 \tl_set:Nn \l_@@_cols_tl { - }
5458 \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).

```

5459 \int_zero_new:N \l_@@_color_int
5460 \int_set:Nn \l_@@_color_int 1
5461 \bool_if:NT \l_@@_respect_blocks_bool
5462 {

```

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`).

```

5463 \seq_set_eq:NN \l_tmpb_seq \g_@@_pos_of_blocks_seq
5464 \seq_set_filter:Nnn \l_tmpa_seq \l_tmpb_seq
5465 { \@@_not_in_exterior_p:nnnnn ##1 }
5466 }
5467 \pgfpicture
5468 \pgf@relevantforpicturesizefalse

```

`#2` is the list of intervals of rows.

```

5469 \clist_map_inline:nn { #2 }
5470 {
5471 \tl_set:Nn \l_tmpa_tl { ##1 }
5472 \tl_if_in:NnTF \l_tmpa_tl { - }
5473 { \@@_cut_on_hyphen:w ##1 \q_stop }
5474 { \tl_set:Nx \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.

```

5475 \int_set:Nn \l_tmpa_int \l_tmpa_tl
5476 \int_set:Nn \l_@@_color_int
5477 { \bool_if:NTF \l_@@_rowcolors_restart_bool 1 \l_tmpa_tl }
5478 \int_zero_new:N \l_@@_tmpc_int
5479 \int_set:Nn \l_@@_tmpc_int \l_tmpb_tl
5480 \int_do_until:nNnn \l_tmpa_int > \l_@@_tmpc_int
5481 {

```

We will compute in `\l_tmpb_int` the last row of the "block".

```

5482 \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).

```

5483 \bool_if:NT \l_@@_respect_blocks_bool
5484 {
5485 \seq_set_filter:Nnn \l_tmpb_seq \l_tmpa_seq
5486 { \@@_intersect_our_row_p:nnnnn #####1 }
5487 \seq_map_inline:Nn \l_tmpb_seq { \@@_rowcolors_i:nnnnn #####1 }

```

Now, the last row of the block is computed in `\l_tmpb_int`.

```

5488 }
5489 \tl_set:Nx \l_@@_rows_tl
5490 { \int_use:N \l_tmpa_int - \int_use:N \l_tmpb_int }

```

`\l_@@_tmpc_tl` will be the color that we will use.

```

5491 \tl_clear_new:N \l_@@_color_tl
5492 \tl_set:Nx \l_@@_color_tl
5493 {
5494 \@@_color_index:n
5495 {
5496 \int_mod:nn
5497 { \l_@@_color_int - 1 }
5498 { \seq_count:N \l_@@_colors_seq }

```

```

5499         + 1
5500     }
5501 }
5502 \tl_if_empty:NF \l_@@_color_tl
5503 {
5504     \@@_add_to_colors_seq:xx
5505     { \tl_if_blank:nF { #1 } { [ #1 ] } { \l_@@_color_tl } }
5506     { \@@_cartesian_color:nn { \l_@@_rows_tl } { \l_@@_cols_tl } }
5507 }
5508 \int_incr:N \l_@@_color_int
5509 \int_set:Nn \l_tmpa_int { \l_tmpb_int + 1 }
5510 }
5511 }
5512 \endpgfpicture
5513 \group_end:
5514 }

```

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.

```

5515 \cs_new:Npn \@@_color_index:n #1
5516 {
5517     \str_if_eq:eeTF { \seq_item:Nn \l_@@_colors_seq { #1 } } { = }
5518     { \@@_color_index:n { #1 - 1 } }
5519     { \seq_item:Nn \l_@@_colors_seq { #1 } }
5520 }

```

The command `\rowcolors` (available in the `\CodeBefore`) is a specialisation of the most general command `\rowlistcolors`. The last argument, which is a optional argument between square brackets is provided by curryfication.

```

5521 \NewDocumentCommand \@@_rowcolors { 0 { } m m m }
5522 { \@@_rowlistcolors [ #1 ] { #2 } { { #3 } , { #4 } } }

5523 \cs_new_protected:Npn \@@_rowcolors_i:nnnnn #1 #2 #3 #4 #5
5524 {
5525     \int_compare:nNnT { #3 } > \l_tmpb_int
5526     { \int_set:Nn \l_tmpb_int { #3 } }
5527 }

5528 \prg_new_conditional:Nnn \@@_not_in_exterior:nnnnn p
5529 {
5530     \bool_lazy_or:nnTF
5531     { \int_compare_p:nNn { #4 } = \c_zero_int }
5532     { \int_compare_p:nNn { #2 } = { \int_eval:n { \c_jCol + 1 } } }
5533     \prg_return_false:
5534     \prg_return_true:
5535 }

```

The following command return true when the block intersects the row `\l_tmpa_int`.

```

5536 \prg_new_conditional:Nnn \@@_intersect_our_row:nnnnn p
5537 {
5538     \bool_if:nTF
5539     {
5540         \int_compare_p:n { #1 <= \l_tmpa_int }
5541         &&
5542         \int_compare_p:n { \l_tmpa_int <= #3 }
5543     }
5544     \prg_return_true:
5545     \prg_return_false:
5546 }

```

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`).

```

5547 \cs_new_protected:Npn \@@_cartesian_path:n #1
5548 {
5549   \bool_lazy_and:nnT
5550   { ! \seq_if_empty_p:N \l_@@_corners_cells_seq }
5551   { \dim_compare_p:nNn { #1 } = \c_zero_dim }
5552   {
5553     \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
5554     \@@_expand_clist:NN \l_@@_rows_tl \c@iRow
5555   }

```

We begin the loop over the columns.

```

5556   \clist_map_inline:Nn \l_@@_cols_tl
5557   {
5558     \tl_set:Nn \l_tmpa_tl { ##1 }
5559     \tl_if_in:NnTF \l_tmpa_tl { - }
5560     { \@@_cut_on_hyphen:w ##1 \q_stop }
5561     { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
5562     \bool_lazy_or:nnT
5563     { \tl_if_blank_p:V \l_tmpa_tl }
5564     { \str_if_eq_p:Vn \l_tmpa_tl { * } }
5565     { \tl_set:Nn \l_tmpa_tl { 1 } }
5566     \bool_lazy_or:nnT
5567     { \tl_if_blank_p:V \l_tmpb_tl }
5568     { \str_if_eq_p:Vn \l_tmpb_tl { * } }
5569     { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@jCol } }
5570     \int_compare:nNnT \l_tmpb_tl > \c@jCol
5571     { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@jCol } }

```

`\l_@@_tmpc_tl` will contain the number of column.

```

5572     \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl

```

If we decide to provide the commands `\cellcolor`, `\rectanglecolor`, `\rowcolor`, `\columncolor`, `\rowcolors` and `\chessboardcolors` in the code-before of a `\SubMatrix`, we will have to modify the following line, by adding a kind of offset. We will have also some other lines to modify.

```

5573     \@@_qpoint:n { col - \l_tmpa_tl }
5574     \int_compare:nNnTF \l_@@_first_col_int = \l_tmpa_tl
5575     { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
5576     { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
5577     \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
5578     \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }

```

We begin the loop over the rows.

```

5579   \clist_map_inline:Nn \l_@@_rows_tl
5580   {
5581     \tl_set:Nn \l_tmpa_tl { #####1 }
5582     \tl_if_in:NnTF \l_tmpa_tl { - }
5583     { \@@_cut_on_hyphen:w #####1 \q_stop }
5584     { \@@_cut_on_hyphen:w #####1 - #####1 \q_stop }
5585     \tl_if_empty:NT \l_tmpa_tl { \tl_set:Nn \l_tmpa_tl { 1 } }
5586     \tl_if_empty:NT \l_tmpb_tl
5587     { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@iRow } }
5588     \int_compare:nNnT \l_tmpb_tl > \c@iRow
5589     { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@iRow } }

```

Now, the numbers of both rows are in `\l_tmpa_tl` and `\l_tmpb_tl`.

```

5590     \seq_if_in:NxF \l_@@_corners_cells_seq
5591     { \l_tmpa_tl - \l_@@_tmpc_tl }
5592     {
5593       \@@_qpoint:n { row - \int_eval:n { \l_tmpb_tl + 1 } }

```

```

5594         \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
5595         \@@_qpoint:n { row - \l_tmpa_tl }
5596         \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
5597         \pgfsetcornersarced { \pgfpoint { #1 } { #1 } }
5598         \pgfpathrectanglecorners
5599         { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
5600         { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5601     }
5602 }
5603 }
5604 }

```

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`).

```

5605 \cs_new_protected:Npn \@@_cartesian_path: { \@@_cartesian_path:n { 0 pt } }

```

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.

```

5606 \cs_new_protected:Npn \@@_expand_clist:NN #1 #2
5607 {
5608     \clist_set_eq:NN \l_tmpa_clist #1
5609     \clist_clear:N #1
5610     \clist_map_inline:Nn \l_tmpa_clist
5611     {
5612         \tl_set:Nn \l_tmpa_tl { ##1 }
5613         \tl_if_in:NnTF \l_tmpa_tl { - }
5614         { \@@_cut_on_hyphen:w ##1 \q_stop }
5615         { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
5616         \bool_lazy_or:nnT
5617         { \tl_if_blank_p:V \l_tmpa_tl }
5618         { \str_if_eq_p:Vn \l_tmpa_tl { * } }
5619         { \tl_set:Nn \l_tmpa_tl { 1 } }
5620         \bool_lazy_or:nnT
5621         { \tl_if_blank_p:V \l_tmpb_tl }
5622         { \str_if_eq_p:Vn \l_tmpb_tl { * } }
5623         { \tl_set:Nx \l_tmpb_tl { \int_use:N #2 } }
5624         \int_compare:nNnT \l_tmpb_tl > #2
5625         { \tl_set:Nx \l_tmpb_tl { \int_use:N #2 } }
5626         \int_step_inline:nnn \l_tmpa_tl \l_tmpb_tl
5627         { \clist_put_right:Nn #1 { #####1 } }
5628     }
5629 }

```

When the user uses the key `color-inside`, the following command will be linked to `\cellcolor` in the tabular.

```

5630 \NewDocumentCommand \@@_cellcolor_tabular { 0 { } m }
5631 {
5632     \tl_gput_right:Nx \g_@@_pre_code_before_tl
5633     {

```

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`).

```

5634         \@@_cellcolor [ #1 ] { \exp_not:n { #2 } }
5635         { \int_use:N \c@iRow - \int_use:N \c@jCol }
5636     }
5637     \ignorespaces
5638 }

```

When the user uses the key `color-inside`, the following command will be linked to `\rowcolor` in the tabular.

```

5639 \NewDocumentCommand \@@_rowcolor_tabular { 0 { } m }
5640 {
5641   \tl_gput_right:Nx \g_@@_pre_code_before_tl
5642   {
5643     \@@_rectanglecolor [ #1 ] { \exp_not:n { #2 } }
5644     { \int_use:N \c@iRow - \int_use:N \c@jCol }
5645     { \int_use:N \c@iRow - \exp_not:n { \int_use:N \c@jCol } }
5646   }
5647   \ignorespaces
5648 }

```

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).

```

5649 \NewDocumentCommand { \@@_rowcolors_tabular } { 0 { } m m }
5650 { \@@_rowlistcolors_tabular [ #1 ] { #2 , #3 } }

```

When the user uses the key `color-inside`, the following command will be linked to `\rowlistcolors` in the `tabular`.

```

5651 \NewDocumentCommand { \@@_rowlistcolors_tabular } { 0 { } m 0 { } }
5652 {
5653   \peek_remove_spaces:n
5654   {
5655     \tl_gput_right:Nx \g__nicematrix_pre_code_before_tl
5656     {
5657       \@@_rowlistcolors
5658       [ #1 ] { \int_use:N \c@iRow } { #2 }
5659       [ restart, cols = \int_use:N \c@jCol - , #3 ]
5660     }
5661   }
5662 }

```

```

5663 \NewDocumentCommand \@@_columncolor_preamble { 0 { } m }
5664 {

```

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).

```

5665   \int_compare:nNnT \c@jCol > \g_@@_col_total_int
5666   {

```

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).

```

5667     \tl_gput_left:Nx \g_@@_pre_code_before_tl
5668     {
5669       \exp_not:N \columncolor [ #1 ]
5670       { \exp_not:n { #2 } } { \int_use:N \c@jCol }
5671     }
5672   }
5673 }

```

23 The vertical and horizontal rules

OnlyMainNiceMatrix

We give to the user the possibility to define new types of columns (with `\newcolumnntype` of `array`) for special vertical rules (*e.g.* rules thicker than the standard ones) which will not extend in the potential exterior rows of the array.

We provide the command `\OnlyMainNiceMatrix` in that goal. However, that command must be no-op outside the environments of `nicematrix` (and so the user will be allowed to use the same new type of column in the environments of `nicematrix` and in the standard environments of `array`). That's why we provide first a global definition of `\OnlyMainNiceMatrix`.

```
5674 \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`.

```
5675 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix:n #1
5676 {
5677   \int_compare:nNnTF \l_@@_first_col_int = 0
5678     { \@@_OnlyMainNiceMatrix_i:n { #1 } }
5679     {
5680       \int_compare:nNnTF \c@jCol = 0
5681         {
5682           \int_compare:nNnF \c@iRow = { -1 }
5683             { \int_compare:nNnF \c@iRow = { \l_@@_last_row_int - 1 } { #1 } }
5684           }
5685         { \@@_OnlyMainNiceMatrix_i:n { #1 } }
5686       }
5687   }
```

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.

```
5688 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix_i:n #1
5689 {
5690   \int_compare:nNnF \c@iRow = 0
5691     {
5692       \int_compare:nNnF \c@iRow = \l_@@_last_row_int
5693         {
5694           \int_compare:nNnT \c@jCol > \c_zero_int
5695             { \bool_if:NF \l_@@_in_last_col_bool { #1 } }
5696           }
5697         }
5698   }
```

Remember that `\c@iRow` is not always inferior to `\l_@@_last_row_int` because `\l_@@_last_row_int` may be equal to -2 or -1 (we can't write `\int_compare:nNnT \c@iRow < \l_@@_last_row_int`).

General system for drawing rules

When a command, environment or “subsystem” of `nicematrix` wants to draw a rule, it will write in the internal `\CodeAfter` a command `\@@_vline:n` or `\@@_hline:n`. Both commands take in as argument a list of *key=value* pairs. That list will first be analyzed with the following set of keys. However, unknown keys will be analyzed further with another set of keys.

```
5699 \keys_define:nn { NiceMatrix / Rules }
5700 {
5701   position .int_set:N = \l_@@_position_int ,
5702   position .value_required:n = true ,
5703   start .int_set:N = \l_@@_start_int ,
5704   start .initial:n = 1 ,
5705   end .code:n =
5706     \bool_lazy_or:nnTF
5707       { \tl_if_empty_p:n { #1 } }
5708       { \str_if_eq_p:nn { #1 } { last } }
5709       { \int_set_eq:NN \l_@@_end_int \c@jCol }
5710       { \int_set:Nn \l_@@_end_int { #1 } }
5711 }
```

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.

```

5712 \keys_define:nn { NiceMatrix / RulesBis }
5713   {
5714     multiplicity .int_set:N = \l_@@_multiplicity_int ,
5715     multiplicity .initial:n = 1 ,
5716     dotted .bool_set:N = \l_@@_dotted_bool ,
5717     dotted .initial:n = false ,
5718     dotted .default:n = true ,
5719     color .code:n = \@@_set_CT@arc@:n { #1 } ,
5720     color .value_required:n = true ,
5721     sep-color .code:n = \@@_set_CT@drsc@:n { #1 } ,
5722     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.

```

5723     tikz .tl_set:N = \l_@@_tikz_rule_tl ,
5724     tikz .value_required:n = true ,
5725     tikz .initial:n = ,
5726     total-width .dim_set:N = \l_@@_rule_width_dim ,
5727     total-width .value_required:n = true ,
5728     width .meta:n = { total-width = #1 } ,
5729     unknown .code:n = \@@_error:n { Unknow~key~for~RulesBis }
5730   }

```

The vertical rules

The following command will be executed in the internal `\CodeAfter`. The argument `#1` is a list of `key=value` pairs.

```

5731 \cs_new_protected:Npn \@@_vline:n #1
5732   {

```

The group is for the options.

```

5733     \group_begin:
5734     \int_zero_new:N \l_@@_end_int
5735     \int_set_eq:NN \l_@@_end_int \c@iRow
5736     \keys_set_known:nnN { NiceMatrix / Rules } { #1 } \l_@@_other_keys_tl

```

The following test is for the case where the user does not use all the columns specified in the preamble of the environment (for instance, a preamble of `|c|c|c|` but only two columns used).

```

5737     \int_compare:nNnT \l_@@_position_int < { \c@jCol + 2 }
5738     \@@_vline_i:
5739     \group_end:
5740   }

```

```

5741 \cs_new_protected:Npn \@@_vline_i:
5742   {
5743     \int_zero_new:N \l_@@_local_start_int
5744     \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 row corresponding to a rule to draw, we note its number in `\l_@@_tmpc_tl`.

```

5745     \tl_set:Nx \l_tmpb_tl { \int_eval:n \l_@@_position_int }
5746     \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
5747     \l_tmpa_tl
5748     {

```

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.


```

5749 \bool_gset_true:N \g_tmpa_bool
5750 \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
5751 { \@@_test_vline_in_block:nnnn ##1 }
5752 \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
5753 { \@@_test_vline_in_block:nnnn ##1 }
5754 \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
5755 { \@@_test_vline_in_stroken_block:nnnn ##1 }
5756 \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_v:
5757 \bool_if:NTF \g_tmpa_bool
5758 {
5759 \int_compare:nNnT \l_@@_local_start_int = 0

```

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.

```

5760 { \int_set:Nn \l_@@_local_start_int \l_tmpa_tl }
5761 }
5762 {
5763 \int_compare:nNnT \l_@@_local_start_int > 0
5764 {
5765 \int_set:Nn \l_@@_local_end_int { \l_tmpa_tl - 1 }
5766 \@@_vline_ii:
5767 \int_zero:N \l_@@_local_start_int
5768 }
5769 }
5770 }
5771 \int_compare:nNnT \l_@@_local_start_int > 0
5772 {
5773 \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
5774 \@@_vline_ii:
5775 }
5776 }

```

```

5777 \cs_new_protected:Npn \@@_test_in_corner_v:
5778 {
5779 \int_compare:nNnTF \l_tmpb_tl = { \int_eval:n { \c@jCol + 1 } }
5780 {
5781 \seq_if_in:NxT
5782 \l_@@_corners_cells_seq
5783 { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
5784 { \bool_set_false:N \g_tmpa_bool }
5785 }
5786 {
5787 \seq_if_in:NxT
5788 \l_@@_corners_cells_seq
5789 { \l_tmpa_tl - \l_tmpb_tl }
5790 {
5791 \int_compare:nNnTF \l_tmpb_tl = 1
5792 { \bool_set_false:N \g_tmpa_bool }
5793 {
5794 \seq_if_in:NxT
5795 \l_@@_corners_cells_seq
5796 { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
5797 { \bool_set_false:N \g_tmpa_bool }
5798 }
5799 }
5800 }
5801 }

```

```

5802 \cs_new_protected:Npn \@@_vline_ii:
5803 {
5804 \keys_set:nV { NiceMatrix / RulesBis } \l_@@_other_keys_tl
5805 \bool_if:NTF \l_@@_dotted_bool

```

```

5806 \@@_vline_iv:
5807 {
5808   \tl_if_empty:NTF \l_@@_tikz_rule_tl
5809   \@@_vline_iii:
5810   \@@_vline_v:
5811 }
5812 }

```

First the case of a standard rule: the user has not used the key dotted nor the key tikz.

```

5813 \cs_new_protected:Npn \@@_vline_iii:
5814 {
5815   \pgfpicture
5816   \pgfrememberpicturepositiononpagetrue
5817   \pgf@relevantforpicturesizefalse
5818   \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
5819   \dim_set_eq:NN \l_tmpa_dim \pgf@y
5820   \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
5821   \dim_set:Nn \l_tmpb_dim
5822   {
5823     \pgf@x
5824     - 0.5 \l_@@_rule_width_dim
5825     +
5826     ( \arrayrulewidth * \l_@@_multiplicity_int
5827       + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
5828   }
5829   \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
5830   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
5831   \bool_lazy_all:nT
5832   {
5833     { \int_compare_p:nNn \l_@@_multiplicity_int > 1 }
5834     { \cs_if_exist_p:N \CT@drsc@ }
5835     { ! \tl_if_blank_p:V \CT@drsc@ }
5836   }
5837   {
5838     \group_begin:
5839     \CT@drsc@
5840     \dim_add:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }
5841     \dim_sub:Nn \l_@@_tmpc_dim { 0.5 \arrayrulewidth }
5842     \dim_set:Nn \l_@@_tmpd_dim
5843     {
5844       \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
5845       * ( \l_@@_multiplicity_int - 1 )
5846     }
5847     \pgfpathrectanglecorners
5848     { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
5849     { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
5850     \pgfusepath { fill }
5851     \group_end:
5852   }
5853   \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
5854   \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
5855   \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
5856   {
5857     \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
5858     \dim_sub:Nn \l_tmpb_dim \doublerulesep
5859     \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
5860     \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
5861   }
5862   \CT@arc@
5863   \pgfsetlinewidth { 1.1 \arrayrulewidth }
5864   \pgfsetrectcap
5865   \pgfusepathqstroke
5866   \endpgfpicture

```

```
5867 }
```

The following code is for the case of a dotted rule (with our system of rounded dots).

```
5868 \cs_new_protected:Npn \@@_vline_iv:
5869 {
5870   \pgfpicture
5871   \pgfrememberpicturepositiononpagetrue
5872   \pgf@relevantforpicturesizefalse
5873   \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
5874   \dim_set:Nn \l_@@_x_initial_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
5875   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
5876   \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
5877   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
5878   \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
5879   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
5880   \CT@arc@
5881   \@@_draw_line:
5882   \endpgfpicture
5883 }
```

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`).

```
5884 \cs_new_protected:Npn \@@_vline_v:
5885 {
5886   \begin {tikzpicture }
5887   \pgfrememberpicturepositiononpagetrue
5888   \pgf@relevantforpicturesizefalse
5889   \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
5890   \dim_set_eq:NN \l_tmpa_dim \pgf@y
5891   \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
5892   \dim_set:Nn \l_tmpb_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
5893   \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
5894   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
5895   \exp_args:NV \tikzset \l_@@_tikz_rule_tl
5896   \use:x { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
5897     ( \l_tmpb_dim , \l_tmpa_dim ) --
5898     ( \l_tmpb_dim , \l_@@_tmpc_dim ) ;
5899   \end { tikzpicture }
5900 }
```

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).

```
5901 \cs_new_protected:Npn \@@_draw_vlines:
5902 {
5903   \int_step_inline:nnn
5904     {
5905       \bool_if:nTF { ! \g_@@_delims_bool && ! \l_@@_except_borders_bool }
5906         1 2
5907     }
5908     {
5909       \bool_if:nTF { ! \g_@@_delims_bool && ! \l_@@_except_borders_bool }
5910         { \int_eval:n { \c@jCol + 1 } }
5911         \c@jCol
5912     }
5913     {
5914       \tl_if_eq:NnF \l_@@_vlines_clist { all }
5915         { \clist_if_in:NnT \l_@@_vlines_clist { ##1 } }
5916         { \@@_vline:n { position = ##1 , total-width = \arrayrulewidth } }
5917     }
5918 }
```

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}`.

```
5919 \cs_new_protected:Npn \@@_hline:n #1
5920 {
```

The group is for the options.

```
5921   \group_begin:
5922   \int_zero_new:N \l_@@_end_int
5923   \int_set_eq:NN \l_@@_end_int \c@jCol
5924   \keys_set_known:nnN { NiceMatrix / Rules } { #1 } \l_@@_other_keys_tl
5925   \@@_hline_i:
5926   \group_end:
5927 }
```

```
5928 \cs_new_protected:Npn \@@_hline_i:
```

```
5929 {
5930   \int_zero_new:N \l_@@_local_start_int
5931   \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`.

```
5932   \tl_set:Nx \l_tmpa_tl { \int_use:N \l_@@_position_int }
5933   \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
5934     \l_tmpb_tl
5935   {
```

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.

```
5936     \bool_gset_true:N \g_tmpa_bool
5937     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
5938       { \@@_test_hline_in_block:nnnn ##1 }
5939     \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
5940       { \@@_test_hline_in_block:nnnn ##1 }
5941     \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
5942       { \@@_test_hline_in_stroken_block:nnnn ##1 }
5943     \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_h:
5944     \bool_if:NTF \g_tmpa_bool
5945       {
5946         \int_compare:nNnT \l_@@_local_start_int = 0
```

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.

```
5947         { \int_set:Nn \l_@@_local_start_int \l_tmpb_tl }
5948       }
5949     {
5950       \int_compare:nNnT \l_@@_local_start_int > 0
5951       {
5952         \int_set:Nn \l_@@_local_end_int { \l_tmpb_tl - 1 }
5953         \@@_hline_ii:
5954         \int_zero:N \l_@@_local_start_int
5955       }
5956     }
5957   }
5958   \int_compare:nNnT \l_@@_local_start_int > 0
5959   {
5960     \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
5961     \@@_hline_ii:
5962   }
5963 }
```

```

5964 \cs_new_protected:Npn \@@_test_in_corner_h:
5965 {
5966   \int_compare:nNnTF \l_tmpa_tl = { \int_eval:n { \c@iRow + 1 } }
5967   {
5968     \seq_if_in:NxT
5969     \l_@@_corners_cells_seq
5970     { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
5971     { \bool_set_false:N \g_tmpa_bool }
5972   }
5973   {
5974     \seq_if_in:NxT
5975     \l_@@_corners_cells_seq
5976     { \l_tmpa_tl - \l_tmpb_tl }
5977     {
5978       \int_compare:nNnTF \l_tmpa_tl = 1
5979       { \bool_set_false:N \g_tmpa_bool }
5980       {
5981         \seq_if_in:NxT
5982         \l_@@_corners_cells_seq
5983         { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
5984         { \bool_set_false:N \g_tmpa_bool }
5985       }
5986     }
5987   }
5988 }

```

```

5989 \cs_new_protected:Npn \@@_hline_ii:
5990 {
5991   \keys_set:nV { NiceMatrix / RulesBis } \l_@@_other_keys_tl
5992   \bool_if:NNTF \l_@@_dotted_bool
5993   \@@_hline_iv:
5994   {
5995     \tl_if_empty:NNTF \l_@@_tikz_rule_tl
5996     \@@_hline_iii:
5997     \@@_hline_v:
5998   }
5999 }

```

First the case of a standard rule (without the keys dotted and tikz).

```

6000 \cs_new_protected:Npn \@@_hline_iii:
6001 {
6002   \pgfpicture
6003   \pgfrememberpicturepositiononpagetrue
6004   \pgf@relevantforpicturesizefalse
6005   \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6006   \dim_set_eq:NN \l_tmpa_dim \pgf@x
6007   \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6008   \dim_set:Nn \l_tmpb_dim
6009   {
6010     \pgf@y
6011     - 0.5 \l_@@_rule_width_dim
6012     +
6013     ( \arrayrulewidth * \l_@@_multiplicity_int
6014       + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6015   }
6016   \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6017   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
6018   \bool_lazy_all:nT
6019   {
6020     { \int_compare_p:nNn \l_@@_multiplicity_int > 1 }
6021     { \cs_if_exist_p:N \CT@drsc@ }
6022     { ! \tl_if_blank_p:V \CT@drsc@ }

```

```

6023     }
6024     {
6025     \group_begin:
6026     \CT@drsc@
6027     \dim_set:Nn \l_@@_tmpd_dim
6028     {
6029     \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
6030     * ( \l_@@_multiplicity_int - 1 )
6031     }
6032     \pgfpathrectanglecorners
6033     { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6034     { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
6035     \pgfusepathqfill
6036     \group_end:
6037     }
6038     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6039     \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
6040     \prg_replicate:n { \l_@@_multiplicity_int - 1 }
6041     {
6042     \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
6043     \dim_sub:Nn \l_tmpb_dim \doublerulesep
6044     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6045     \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
6046     }
6047     \CT@arc@
6048     \pgfsetlinewidth { 1.1 \arrayrulewidth }
6049     \pgfsetrectcap
6050     \pgfusepathqstroke
6051     \endpgfpicture
6052     }

```

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}

```

$$\begin{array}{cccc} \hline 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ \hdottedline 1 & 2 & 3 & 4 \end{array}$$

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}

```

$$\begin{array}{cccc} \hline 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ \hdottedline 1 & 2 & 3 & 4 \end{array}$$

```

6053 \cs_new_protected:Npn \@@_hline_iv:
6054 {
6055     \pgfpicture
6056     \pgfrememberpicturepositiononpagetrue
6057     \pgf@relevantforpicturesizefalse
6058     \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6059     \dim_set:Nn \l_@@_y_initial_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
6060     \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
6061     \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6062     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
6063     \int_compare:nNnT \l_@@_local_start_int = 1
6064     {
6065     \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim

```

```

6066     \bool_if:NF \g_@@_delims_bool
6067     { \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.

```

6068     \tl_if_eq:NnF \g_@@_left_delim_tl (
6069     { \dim_add:Nn \l_@@_x_initial_dim { 0.5 \l_@@_xdots_inter_dim } }
6070     )
6071     \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6072     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
6073     \int_compare:nNnT \l_@@_local_end_int = \c@jCol
6074     {
6075     \dim_add:Nn \l_@@_x_final_dim \l_@@_right_margin_dim
6076     \bool_if:NF \g_@@_delims_bool
6077     { \dim_add:Nn \l_@@_x_final_dim \arraycolsep }
6078     \tl_if_eq:NnF \g_@@_right_delim_tl )
6079     { \dim_gsub:Nn \l_@@_x_final_dim { 0.5 \l_@@_xdots_inter_dim } }
6080     )
6081     \CT@arc@
6082     \@@_draw_line:
6083     \endpgfpicture
6084     }

```

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`).

```

6085 \cs_new_protected:Npn \@@_hline_v:
6086 {
6087   \begin { tikzpicture }
6088   \pgfrememberpicturepositiononpagetrue
6089   \pgf@relevantforpicturesizefalse
6090   \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6091   \dim_set_eq:NN \l_tmpa_dim \pgf@x
6092   \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6093   \dim_set:Nn \l_tmpb_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
6094   \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6095   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
6096   \exp_args:NV \tikzset \l_@@_tikz_rule_tl
6097   \use:x { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6098     ( \l_tmpa_dim , \l_tmpb_dim ) --
6099     ( \l_@@_tmpc_dim , \l_tmpb_dim ) ;
6100   \end { tikzpicture }
6101 }

```

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).

```

6102 \cs_new_protected:Npn \@@_draw_hlines:
6103 {
6104   \int_step_inline:nnn
6105     {
6106     \bool_if:nTF { ! \g_@@_delims_bool && ! \l_@@_except_borders_bool }
6107     1 2
6108     }
6109     {
6110     \bool_if:nTF { ! \g_@@_delims_bool && ! \l_@@_except_borders_bool }
6111     { \int_eval:n { \c@iRow + 1 } }
6112     \c@iRow
6113     }
6114     {
6115     \tl_if_eq:NnF \l_@@_hlines_clist { all }
6116     { \clist_if_in:NnT \l_@@_hlines_clist { ##1 } }
6117     { \@@_hline:n { position = ##1 , total-width = \arrayrulewidth } }

```

```

6118     }
6119 }

```

The command `\@@_Hline:` will be linked to `\Hline` in the environments of `nicematrix`.

```

6120 \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.

```

6121 \cs_set:Npn \@@_Hline_i:n #1
6122 {
6123   \peek_remove_spaces:n
6124   {
6125     \peek_meaning:NTF \Hline
6126     { \@@_Hline_ii:nn { #1 + 1 } }
6127     { \@@_Hline_iii:n { #1 } }
6128   }
6129 }

6130 \cs_set:Npn \@@_Hline_ii:nn #1 #2 { \@@_Hline_i:n { #1 } }

6131 \cs_set:Npn \@@_Hline_iii:n #1
6132 {
6133   \peek_meaning:NTF [
6134   { \@@_Hline_iv:nw { #1 } }
6135   { \@@_Hline_iv:nw { #1 } [ ] }
6136 }

6137 \cs_set:Npn \@@_Hline_iv:nw #1 [ #2 ]
6138 {
6139   \@@_compute_rule_width:n { multiplicity = #1 , #2 }
6140   \skip_vertical:n { \l_@@_rule_width_dim }
6141   \tl_gput_right:Nx \g_@@_pre_code_after_tl
6142   {
6143     \@@_hline:n
6144     {
6145       multiplicity = #1 ,
6146       position = \int_eval:n { \c@iRow + 1 } ,
6147       total-width = \dim_use:N \l_@@_rule_width_dim ,
6148       #2
6149     }
6150   }
6151   \egroup
6152 }

```

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.

Among the keys available in that list, there is the key `letter` to specify a letter that the final user will use in the preamble of the array. All the letters defined by this way by the final user for such customized rules are added in the set of keys `{NiceMatrix / ColumnTypes}`. That set of keys is used to store the characteristics of those types of rules for convenience: the keys of that set of keys won't never be used as keys by the final user (he will use, instead, letters in the preamble of its array).

```

6153 \keys_define:nn { NiceMatrix / ColumnTypes } { }

```

The following command will create the customized rule (it is executed when the final user uses the key `custom-line`, for example in `\NiceMatrixOptions`).

```

6154 \cs_new_protected:Npn \@@_custom_line:n #1
6155 {
6156   \str_clear_new:N \l_@@_command_str
6157   \str_clear_new:N \l_@@_ccommand_str
6158   \str_clear_new:N \l_@@_letter_str
6159   \tl_clear_new:N \l_@@_other_keys_tl
6160   \keys_set_known:nn { 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.

```

6161 \bool_lazy_all:nTF
6162 {
6163   { \str_if_empty_p:N \l_@@_letter_str }
6164   { \str_if_empty_p:N \l_@@_command_str }
6165   { \str_if_empty_p:N \l_@@_ccommand_str }
6166 }
6167 { \@@_error:n { No~letter~and~no~command } }
6168 { \exp_args:NV \@@_custom_line_i:n \l_@@_other_keys_tl }
6169 }
6170 \keys_define:nn { NiceMatrix / custom-line }
6171 {
6172   letter .str_set:N = \l_@@_letter_str ,
6173   letter .value_required:n = true ,
6174   command .str_set:N = \l_@@_command_str ,
6175   command .value_required:n = true ,
6176   ccommand .str_set:N = \l_@@_ccommand_str ,
6177   ccommand .value_required:n = true ,
6178 }

```

```

6179 \cs_new_protected:Npn \@@_custom_line_i:n #1
6180 {

```

The following flags will be raised when the keys `tikz`, `dotted` and `color` are used (in the `custom-line`).

```

6181 \bool_set_false:N \l_@@_tikz_rule_bool
6182 \bool_set_false:N \l_@@_dotted_rule_bool
6183 \bool_set_false:N \l_@@_color_bool
6184 \keys_set:nn { NiceMatrix / custom-line-bis } { #1 }
6185 \bool_if:NT \l_@@_tikz_rule_bool
6186 {
6187   \IfPackageLoadedTF { tikz }
6188   { }
6189   { \@@_error:n { tikz~in~custom~line~without~tikz } }
6190   \bool_if:NT \l_@@_color_bool
6191   { \@@_error:n { color~in~custom~line~with~tikz } }
6192 }
6193 \bool_if:nTF
6194 {
6195   \int_compare_p:nNn \l_@@_multiplicity_int > 1
6196   && \l_@@_dotted_rule_bool
6197 }
6198 { \@@_error:n { key~multiplicity~with~dotted } }
6199 \str_if_empty:NF \l_@@_letter_str
6200 {
6201   \int_compare:nTF { \str_count:N \l_@@_letter_str != 1 }
6202   { \@@_error:n { Several~letters } }
6203   {
6204     \exp_args:NnV \tl_if_in:NnTF
6205     \c_@@_forbidden_letters_str \l_@@_letter_str
6206     { \@@_error:n { Forbidden~letter } }
6207     {

```

The final user can, locally, redefine a letter of column type. That's compatible with the use of `\keys_define:nn`: the definition is local and may overwrite a previous definition.

```

6208 \keys_define:nx { NiceMatrix / ColumnTypes }
6209 {
6210   \l_@@_letter_str .code:n =

```

```

6211             { \l_@@_v_custom_line:n { \exp_not:n { #1 } } }
6212         }
6213     }
6214 }
6215 }
6216 \str_if_empty:NF \l_@@_command_str { \l_@@_h_custom_line:n { #1 } }
6217 \str_if_empty:NF \l_@@_ccommand_str { \l_@@_c_custom_line:n { #1 } }
6218 }
6219 \str_const:Nn \c_@@_forbidden_letters_str { lcrpmbVX|()[]!@<> }

```

The previous command `\l_@@_custom_line_i:n` uses the following set of keys. However, the whole definition of the customized lines (as provided by the final user as argument of `custom-line`) will also be used further with other sets of keys (for instance `{NiceMatrix/Rules}`). That's why the following set of keys has some keys which are no-op.

```

6220 \keys_define:nn { NiceMatrix / custom-line-bis }
6221 {
6222     multiplicity .int_set:N = \l_@@_multiplicity_int ,
6223     multiplicity .initial:n = 1 ,
6224     multiplicity .value_required:n = true ,
6225     color .code:n = \bool_set_true:N \l_@@_color_bool ,
6226     color .value_required:n = true ,
6227     tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6228     tikz .value_required:n = true ,
6229     dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6230     dotted .value_forbidden:n = true ,
6231     total-width .code:n = { } ,
6232     total-width .value_required:n = true ,
6233     width .code:n = { } ,
6234     width .value_required:n = true ,
6235     sep-color .code:n = { } ,
6236     sep-color .value_required:n = true ,
6237     unknown .code:n = \@@_error:n { Unknown~key~for~custom~line }
6238 }

```

The following keys will indicate whether the keys `dotted`, `tikz` and `color` are used in the use of a `custom-line`.

```

6239 \bool_new:N \l_@@_dotted_rule_bool
6240 \bool_new:N \l_@@_tikz_rule_bool
6241 \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`.

```

6242 \keys_define:nn { NiceMatrix / custom-line-width }
6243 {
6244     multiplicity .int_set:N = \l_@@_multiplicity_int ,
6245     multiplicity .initial:n = 1 ,
6246     multiplicity .value_required:n = true ,
6247     tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6248     total-width .code:n = \dim_set:Nn \l_@@_rule_width_dim { #1 }
6249         \bool_set_true:N \l_@@_total_width_bool ,
6250     total-width .value_required:n = true ,
6251     width .meta:n = { total-width = #1 } ,
6252     dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6253 }

```

The following command will create the command that the final user will use in its array to draw an horizontal rule (hence the 'h' in the name) with the full width of the array. `#1` is the whole set of keys to pass to the command `\l_@@_hline:n` (which is in the internal `\CodeAfter`).

```

6254 \cs_new_protected:Npn \l_@@_h_custom_line:n #1
6255 {

```

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`.

```

6256   \cs_set:cpn { nicematrix - \l_@@_command_str }
6257     {
6258       \noalign
6259         {
6260           \@_compute_rule_width:n { #1 }
6261           \skip_vertical:n { \l_@@_rule_width_dim }
6262           \tl_gput_right:Nx \g_@@_pre_code_after_tl
6263             {
6264               \@_hline:n
6265                 {
6266                   #1 ,
6267                   position = \int_eval:n { \c@iRow + 1 } ,
6268                   total-width = \dim_use:N \l_@@_rule_width_dim
6269                 }
6270             }
6271         }
6272     }
6273   \seq_put_left:NV \l_@@_custom_line_commands_seq \l_@@_command_str
6274 }

```

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`).

```

6275 \cs_new_protected:Npn \@@_c_custom_line:n #1
6276   {

```

Here, we need an expandable command since it begins with an `\noalign`.

```

6277   \exp_args:Nc \NewExpandableDocumentCommand
6278     { nicematrix - \l_@@_ccommand_str }
6279     { 0 { } m }
6280     {
6281       \noalign
6282         {
6283           \@_compute_rule_width:n { #1 , ##1 }
6284           \skip_vertical:n { \l_@@_rule_width_dim }
6285           \clist_map_inline:nn
6286             { ##2 }
6287             { \@@_c_custom_line_i:nn { #1 , ##1 } { ####1 } }
6288         }
6289     }
6290   \seq_put_left:NV \l_@@_custom_line_commands_seq \l_@@_ccommand_str
6291 }

```

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`.

```

6292 \cs_new_protected:Npn \@@_c_custom_line_i:nn #1 #2
6293   {
6294     \str_if_in:nnTF { #2 } { - }
6295       { \@@_cut_on_hyphen:w #2 \q_stop }
6296       { \@@_cut_on_hyphen:w #2 - #2 \q_stop }
6297     \tl_gput_right:Nx \g_@@_pre_code_after_tl
6298       {
6299         \@_hline:n
6300           {
6301             #1 ,
6302             start = \l_tmpa_tl ,
6303             end = \l_tmpb_tl ,
6304             position = \int_eval:n { \c@iRow + 1 } ,
6305             total-width = \dim_use:N \l_@@_rule_width_dim
6306           }

```

```

6307     }
6308   }
6309 \cs_new_protected:Npn \@@_compute_rule_width:n #1
6310   {
6311     \bool_set_false:N \l_@@_tikz_rule_bool
6312     \bool_set_false:N \l_@@_total_width_bool
6313     \bool_set_false:N \l_@@_dotted_rule_bool
6314     \keys_set_known:n { NiceMatrix / custom-line-width } { #1 }
6315     \bool_if:NF \l_@@_total_width_bool
6316     {
6317       \bool_if:NTF \l_@@_dotted_rule_bool
6318         { \dim_set:Nn \l_@@_rule_width_dim { 2 \l_@@_xdots_radius_dim } }
6319         {
6320           \bool_if:NF \l_@@_tikz_rule_bool
6321           {
6322             \dim_set:Nn \l_@@_rule_width_dim
6323             {
6324               \arrayrulewidth * \l_@@_multiplicity_int
6325               + \doublerulesep * ( \l_@@_multiplicity_int - 1 )
6326             }
6327           }
6328         }
6329     }
6330   }
6331 \cs_new_protected:Npn \@@_v_custom_line:n #1
6332   {
6333     \@@_compute_rule_width:n { #1 }

```

In the following line, the `\dim_use:N` is mandatory since we do an expansion.

```

6334   \tl_gput_right:Nx \g_@@_preamble_tl
6335     { \exp_not:N ! { \skip_horizontal:n { \dim_use:N \l_@@_rule_width_dim } } }
6336   \tl_gput_right:Nx \g_@@_pre_code_after_tl
6337     {
6338       \@@_vline:n
6339       {
6340         #1 ,
6341         position = \int_eval:n { \c@jCol + 1 } ,
6342         total-width = \dim_use:N \l_@@_rule_width_dim
6343       }
6344     }
6345   }
6346 \@@_custom_line:n
6347   { 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.

```

6348 \cs_new_protected:Npn \@@_test_hline_in_block:nnnnn #1 #2 #3 #4 #5
6349   {
6350     \bool_lazy_all:nT
6351     {
6352       { \int_compare_p:nNn \l_tmpa_tl > { #1 } }
6353       { \int_compare_p:nNn \l_tmpa_tl < { #3 + 1 } }
6354       { \int_compare_p:nNn \l_tmpb_tl > { #2 - 1 } }
6355       { \int_compare_p:nNn \l_tmpb_tl < { #4 + 1 } }
6356     }
6357     { \bool_gset_false:N \g_tmpa_bool }
6358   }

```

The same for vertical rules.

```

6359 \cs_new_protected:Npn \@_test_vline_in_block:nnnn #1 #2 #3 #4 #5
6360 {
6361   \bool_lazy_all:nT
6362     {
6363       { \int_compare_p:nNn \l_tmpa_tl > { #1 - 1 } }
6364       { \int_compare_p:nNn \l_tmpa_tl < { #3 + 1 } }
6365       { \int_compare_p:nNn \l_tmpb_tl > { #2 } }
6366       { \int_compare_p:nNn \l_tmpb_tl < { #4 + 1 } }
6367     }
6368     { \bool_gset_false:N \g_tmpa_bool }
6369 }

6370 \cs_new_protected:Npn \@_test_hline_in_stroken_block:nnnn #1 #2 #3 #4
6371 {
6372   \bool_lazy_all:nT
6373     {
6374       {
6375         ( \int_compare_p:nNn \l_tmpa_tl = { #1 } )
6376         || ( \int_compare_p:nNn \l_tmpa_tl = { #3 + 1 } )
6377       }
6378       { \int_compare_p:nNn \l_tmpb_tl > { #2 - 1 } }
6379       { \int_compare_p:nNn \l_tmpb_tl < { #4 + 1 } }
6380     }
6381     { \bool_gset_false:N \g_tmpa_bool }
6382 }

6383 \cs_new_protected:Npn \@_test_vline_in_stroken_block:nnnn #1 #2 #3 #4
6384 {
6385   \bool_lazy_all:nT
6386     {
6387       { \int_compare_p:nNn \l_tmpa_tl > { #1 - 1 } }
6388       { \int_compare_p:nNn \l_tmpa_tl < { #3 + 1 } }
6389     }
6390     {
6391       ( \int_compare_p:nNn \l_tmpb_tl = { #2 } )
6392       || ( \int_compare_p:nNn \l_tmpb_tl = { #4 + 1 } )
6393     }
6394     { \bool_gset_false:N \g_tmpa_bool }
6395 }

```

24 The key corners

When the key `corners` is raised, the rules are not drawn in the corners. Of course, we have to compute the corners before we begin to draw the rules.

```

6396 \cs_new_protected:Npn \@_compute_corners:
6397 {

```

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.

```

6398   \seq_clear_new:N \l_@@_corners_cells_seq
6399   \clist_map_inline:Nn \l_@@_corners_clist
6400     {
6401       \str_case:nnF { ##1 }
6402         {
6403           { NW }
6404           { \@@_compute_a_corner:nnnnn 1 1 1 1 \c@iRow \c@jCol }
6405           { NE }
6406           { \@@_compute_a_corner:nnnnn 1 \c@jCol 1 { -1 } \c@iRow 1 }
6407           { SW }

```

```

6408         { \@@_compute_a_corner:nnnnnn \c@iRow 1 { -1 } 1 1 \c@jCol }
6409         { SE }
6410         { \@@_compute_a_corner:nnnnnn \c@iRow \c@jCol { -1 } { -1 } 1 1 }
6411     }
6412     { \@@_error:nn { bad~corner } { ##1 } }
6413 }

```

Even if the user has used the key `corners` the list of cells in the corners may be empty.

```

6414     \seq_if_empty:NF \l_@@_corners_cells_seq
6415     {

```

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.

```

6416         \tl_gput_right:Nx \g_@@_aux_tl
6417         {
6418             \seq_set_from_clist:Nn \exp_not:N \l_@@_corners_cells_seq
6419             { \seq_use:Nnnn \l_@@_corners_cells_seq , , , }
6420         }
6421     }
6422 }

```

“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.

```

6423 \cs_new_protected:Npn \@@_compute_a_corner:nnnnnn #1 #2 #3 #4 #5 #6
6424 {

```

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.

```

6425     \bool_set_false:N \l_tmpa_bool
6426     \int_zero_new:N \l_@@_last_empty_row_int
6427     \int_set:Nn \l_@@_last_empty_row_int { #1 }
6428     \int_step_inline:nnnn { #1 } { #3 } { #5 }
6429     {
6430         \@@_test_if_cell_in_a_block:nn { ##1 } { \int_eval:n { #2 } }
6431         \bool_lazy_or:nnTF
6432         {
6433             \cs_if_exist_p:c
6434             { pgf @ sh @ ns @ \@@_env: - ##1 - \int_eval:n { #2 } }
6435         }
6436         \l_tmpb_bool
6437         { \bool_set_true:N \l_tmpa_bool }
6438         {
6439             \bool_if:NF \l_tmpa_bool
6440             { \int_set:Nn \l_@@_last_empty_row_int { ##1 } }
6441         }
6442     }

```

Now, you determine the last empty cell in the row of number 1.

```

6443 \bool_set_false:N \l_tmpa_bool
6444 \int_zero_new:N \l_@@_last_empty_column_int
6445 \int_set:Nn \l_@@_last_empty_column_int { #2 }
6446 \int_step_inline:nnnn { #2 } { #4 } { #6 }
6447 {
6448   \@@_test_if_cell_in_a_block:nn { \int_eval:n { #1 } } { ##1 }
6449   \bool_lazy_or:nnTF
6450     \l_tmpb_bool
6451     {
6452       \cs_if_exist_p:c
6453         { pgf @ sh @ ns @ \@@_env: - \int_eval:n { #1 } - ##1 }
6454     }
6455     { \bool_set_true:N \l_tmpa_bool }
6456     {
6457       \bool_if:NF \l_tmpa_bool
6458         { \int_set:Nn \l_@@_last_empty_column_int { ##1 } }
6459     }
6460 }

```

Now, we loop over the rows.

```

6461 \int_step_inline:nnnn { #1 } { #3 } \l_@@_last_empty_row_int
6462 {

```

We treat the row number ##1 with another loop.

```

6463   \bool_set_false:N \l_tmpa_bool
6464   \int_step_inline:nnnn { #2 } { #4 } \l_@@_last_empty_column_int
6465   {
6466     \@@_test_if_cell_in_a_block:nn { ##1 } { ##### }
6467     \bool_lazy_or:nnTF
6468       \l_tmpb_bool
6469       {
6470         \cs_if_exist_p:c
6471           { pgf @ sh @ ns @ \@@_env: - ##1 - ##### }
6472       }
6473       { \bool_set_true:N \l_tmpa_bool }
6474       {
6475         \bool_if:NF \l_tmpa_bool
6476         {
6477           \int_set:Nn \l_@@_last_empty_column_int { ##### }
6478           \seq_put_right:Nn
6479             \l_@@_corners_cells_seq
6480             { ##1 - ##### }
6481         }
6482       }
6483     }
6484   }
6485 }

```

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`).

```

6486 \cs_new_protected:Npn \@@_test_if_cell_in_a_block:nn #1 #2
6487 {
6488   \int_set:Nn \l_tmpa_int { #1 }
6489   \int_set:Nn \l_tmpb_int { #2 }
6490   \bool_set_false:N \l_tmpb_bool
6491   \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6492     { \@@_test_if_cell_in_block:nnnnnn \l_tmpa_int \l_tmpb_int ##1 }
6493 }
6494 \cs_new_protected:Npn \@@_test_if_cell_in_block:nnnnnn #1 #2 #3 #4 #5 #6 #7
6495 {
6496   \int_compare:nNnT { #3 } < { \int_eval:n { #1 + 1 } }

```

```

6497 {
6498   \int_compare:nNnT { #1 } < { \int_eval:n { #5 + 1 } }
6499   {
6500     \int_compare:nNnT { #4 } < { \int_eval:n { #2 + 1 } }
6501     {
6502       \int_compare:nNnT { #2 } < { \int_eval:n { #6 + 1 } }
6503       { \bool_set_true:N \l_tmpb_bool }
6504     }
6505   }
6506 }
6507 }

```

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.

```

6508 \bool_new:N \l_@@_block_auto_columns_width_bool

```

Up to now, there is only one option available for the environment {NiceMatrixBlock}.

```

6509 \keys_define:nn { NiceMatrix / NiceMatrixBlock }
6510 {
6511   auto-columns-width .code:n =
6512   {
6513     \bool_set_true:N \l_@@_block_auto_columns_width_bool
6514     \dim_gzero_new:N \g_@@_max_cell_width_dim
6515     \bool_set_true:N \l_@@_auto_columns_width_bool
6516   }
6517 }

6518 \NewDocumentEnvironment { NiceMatrixBlock } { ! 0 { } }
6519 {
6520   \int_gincr:N \g_@@_NiceMatrixBlock_int
6521   \dim_zero:N \l_@@_columns_width_dim
6522   \keys_set:nn { NiceMatrix / NiceMatrixBlock } { #1 }
6523   \bool_if:NT \l_@@_block_auto_columns_width_bool
6524   {
6525     \cs_if_exist:cT { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
6526     {
6527       \exp_args:NNc \dim_set:Nn \l_@@_columns_width_dim
6528       { @@_max_cell_width _ \int_use:N \g_@@_NiceMatrixBlock_int }
6529     }
6530   }
6531 }

```

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).

```

6532 {
6533   \bool_if:NT \l_@@_block_auto_columns_width_bool
6534   {
6535     \iow_shipout:Nn \@mainaux \ExplSyntaxOn
6536     \iow_shipout:Nx \@mainaux
6537     {
6538       \cs_gset:cpn
6539       { @@ _ 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.

```

6540         { \dim_eval:n { \g_@@_max_cell_width_dim + \arrayrulewidth } }
6541     }
6542     \iow_shipout:Nn \@mainaux \ExplSyntaxOff
6543 }
6544 \ignorespacesafterend
6545 }

```

26 The extra nodes

First, two variants of the functions `\dim_min:nn` and `\dim_max:nn`.

```

6546 \cs_generate_variant:Nn \dim_min:nn { v n }
6547 \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).

```

6548 \cs_new_protected:Npn \@@_create_extra_nodes:
6549 {
6550     \bool_if:nTF \l_@@_medium_nodes_bool
6551     {
6552         \bool_if:NTF \l_@@_large_nodes_bool
6553         \@@_create_medium_and_large_nodes:
6554         \@@_create_medium_nodes:
6555     }
6556     { \bool_if:NT \l_@@_large_nodes_bool \@@_create_large_nodes: }
6557 }

```

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`.

```

6558 \cs_new_protected:Npn \@@_computations_for_medium_nodes:
6559 {
6560     \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6561     {
6562         \dim_zero_new:c { l_@@_row_\@@_i: _min_dim }
6563         \dim_set_eq:cN { l_@@_row_\@@_i: _min_dim } \c_max_dim
6564         \dim_zero_new:c { l_@@_row_\@@_i: _max_dim }
6565         \dim_set:cn { l_@@_row_\@@_i: _max_dim } { - \c_max_dim }
6566     }
6567     \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
6568     {
6569         \dim_zero_new:c { l_@@_column_\@@_j: _min_dim }

```

```

6570     \dim_set_eq:cN { l_@@_column_\@@_j: _min_dim } \c_max_dim
6571     \dim_zero_new:c { l_@@_column_\@@_j: _max_dim }
6572     \dim_set:cn { l_@@_column_\@@_j: _max_dim } { - \c_max_dim }
6573 }

```

We begin the two nested loops over the rows and the columns of the array.

```

6574     \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6575     {
6576         \int_step_variable:nnNn
6577         \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.

```

6578         {
6579             \cs_if_exist:cT
6580             { 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`.

```

6581         {
6582             \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { south-west }
6583             \dim_set:cn { l_@@_row_\@@_i: _min_dim }
6584             { \dim_min:vn { l_@@_row _ \@@_i: _min_dim } \pgf@y }
6585             \seq_if_in:NxF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
6586             {
6587                 \dim_set:cn { l_@@_column _ \@@_j: _min_dim }
6588                 { \dim_min:vn { l_@@_column _ \@@_j: _min_dim } \pgf@x }
6589             }

```

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`.

```

6590             \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { north-east }
6591             \dim_set:cn { l_@@_row _ \@@_i: _ max_dim }
6592             { \dim_max:vn { l_@@_row _ \@@_i: _ max_dim } \pgf@y }
6593             \seq_if_in:NxF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
6594             {
6595                 \dim_set:cn { l_@@_column _ \@@_j: _ max_dim }
6596                 { \dim_max:vn { l_@@_column _ \@@_j: _ max_dim } \pgf@x }
6597             }
6598         }
6599     }
6600 }

```

Now, we have to deal with empty rows or empty columns since we don't have created nodes in such rows and columns.

```

6601     \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6602     {
6603         \dim_compare:nNnT
6604         { \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } } = \c_max_dim
6605         {
6606             \@@_qpoint:n { row - \@@_i: - base }
6607             \dim_set:cn { l_@@_row _ \@@_i: _ max _ dim } \pgf@y
6608             \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim } \pgf@y
6609         }
6610     }
6611     \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
6612     {
6613         \dim_compare:nNnT
6614         { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } } = \c_max_dim
6615         {
6616             \@@_qpoint:n { col - \@@_j: }
6617             \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim } \pgf@y
6618             \dim_set:cn { l_@@_column _ \@@_j: _ min _ dim } \pgf@y
6619         }
6620     }
6621 }

```

Here is the command `\@@_create_medium_nodes:`. When this command is used, the “medium nodes” are created.

```
6622 \cs_new_protected:Npn \@@_create_medium_nodes:
6623   {
6624     \pgfpicture
6625       \pgfrememberpicturepositiononpagetrue
6626       \pgf@relevantforpicturesizefalse
6627       \@@_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”.

```
6628     \tl_set:Nn \l_@@_suffix_tl { -medium }
6629     \@@_create_nodes:
6630     \endpgfpicture
6631   }
```

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:`.

```
6632 \cs_new_protected:Npn \@@_create_large_nodes:
6633   {
6634     \pgfpicture
6635       \pgfrememberpicturepositiononpagetrue
6636       \pgf@relevantforpicturesizefalse
6637       \@@_computations_for_medium_nodes:
6638       \@@_computations_for_large_nodes:
6639       \tl_set:Nn \l_@@_suffix_tl { - large }
6640       \@@_create_nodes:
6641     \endpgfpicture
6642   }

6643 \cs_new_protected:Npn \@@_create_medium_and_large_nodes:
6644   {
6645     \pgfpicture
6646       \pgfrememberpicturepositiononpagetrue
6647       \pgf@relevantforpicturesizefalse
6648       \@@_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”.

```
6649     \tl_set:Nn \l_@@_suffix_tl { - medium }
6650     \@@_create_nodes:
6651     \@@_computations_for_large_nodes:
6652     \tl_set:Nn \l_@@_suffix_tl { - large }
6653     \@@_create_nodes:
6654     \endpgfpicture
6655   }
```

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.

```
6656 \cs_new_protected:Npn \@@_computations_for_large_nodes:
6657   {
6658     \int_set:Nn \l_@@_first_row_int 1
6659     \int_set:Nn \l_@@_first_col_int 1
```

¹⁴If we want to create both, we have to use `\@@_create_medium_and_large_nodes:`

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`.

```

6660 \int_step_variable:nNn { \c@iRow - 1 } \@@_i:
6661 {
6662   \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim }
6663   {
6664     (
6665       \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } +
6666       \dim_use:c { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
6667     )
6668     / 2
6669   }
6670   \dim_set_eq:cc { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
6671   { l_@@_row_\@@_i: _ min_dim }
6672 }
6673 \int_step_variable:nNn { \c@jCol - 1 } \@@_j:
6674 {
6675   \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim }
6676   {
6677     (
6678       \dim_use:c { l_@@_column _ \@@_j: _ max _ dim } +
6679       \dim_use:c
6680         { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
6681     )
6682     / 2
6683   }
6684   \dim_set_eq:cc { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
6685   { l_@@_column _ \@@_j: _ max _ dim }
6686 }

```

Here, we have to use `\dim_sub:cn` because of the number 1 in the name.

```

6687 \dim_sub:cn
6688 { l_@@_column _ 1 _ min _ dim }
6689 \l_@@_left_margin_dim
6690 \dim_add:cn
6691 { l_@@_column _ \int_use:N \c@jCol _ max _ dim }
6692 \l_@@_right_margin_dim
6693 }

```

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).

```

6694 \cs_new_protected:Npn \@@_create_nodes:
6695 {
6696   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6697   {
6698     \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
6699     {

```

We draw the rectangular node for the cell (`\@@_i-\@@_j`).

```

6700 \@@_pgf_rect_node:nnnnn
6701 { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
6702 { \dim_use:c { l_@@_column _ \@@_j: _ min_dim } }
6703 { \dim_use:c { l_@@_row _ \@@_i: _ min_dim } }
6704 { \dim_use:c { l_@@_column _ \@@_j: _ max_dim } }
6705 { \dim_use:c { l_@@_row _ \@@_i: _ max_dim } }
6706 \str_if_empty:NF \l_@@_name_str
6707 {
6708   \pgfnodealias
6709   { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }

```

```

6710         { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
6711     }
6712 }
6713 }

```

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 .

```

6714     \cs_if_exist_use:NF
6715         \seq_map_pairwise_function:NNN
6716         \seq_mapthread_function:NNN
6717         \g_@@_multicolumn_cells_seq
6718         \g_@@_multicolumn_sizes_seq
6719         \@@_node_for_multicolumn:nn
6720     }

6721 \cs_new_protected:Npn \@@_extract_coords_values: #1 - #2 \q_stop
6722 {
6723     \cs_set_nopar:Npn \@@_i: { #1 }
6724     \cs_set_nopar:Npn \@@_j: { #2 }
6725 }

```

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”).

```

6726 \cs_new_protected:Npn \@@_node_for_multicolumn:nn #1 #2
6727 {
6728     \@@_extract_coords_values: #1 \q_stop
6729     \@@_pgf_rect_node:nnnnn
6730     { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
6731     { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } }
6732     { \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } }
6733     { \dim_use:c { l_@@_column _ \int_eval:n { \@@_j: +#2-1 } _ max _ dim } }
6734     { \dim_use:c { l_@@_row _ \@@_i: _ max _ dim } }
6735     \str_if_empty:NF \l_@@_name_str
6736     {
6737         \pgfnodealias
6738         { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
6739         { \int_use:N \g_@@_env_int - \@@_i: - \@@_j: \l_@@_suffix_tl}
6740     }
6741 }

```

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.

```

6742 \keys_define:nn { NiceMatrix / Block / FirstPass }
6743 {
6744     l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
6745     l .value_forbidden:n = true ,
6746     r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
6747     r .value_forbidden:n = true ,
6748     c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
6749     c .value_forbidden:n = true ,
6750     L .code:n = \str_set:Nn \l_@@_hpos_block_str l ,

```

```

6751 L .value_forbidden:n = true ,
6752 R .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
6753 R .value_forbidden:n = true ,
6754 C .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
6755 C .value_forbidden:n = true ,
6756 t .code:n = \str_set:Nn \l_@@_vpos_of_block_str t ,
6757 t .value_forbidden:n = true ,
6758 T .code:n = \str_set:Nn \l_@@_vpos_of_block_str T ,
6759 T .value_forbidden:n = true ,
6760 b .code:n = \str_set:Nn \l_@@_vpos_of_block_str b ,
6761 b .value_forbidden:n = true ,
6762 B .code:n = \str_set:Nn \l_@@_vpos_of_block_str B ,
6763 B .value_forbidden:n = true ,
6764 color .code:n =
6765   \@@_color:n { #1 }
6766   \tl_set_rescan:Nnn
6767     \l_@@_draw_tl
6768     { \char_set_catcode_other:N ! }
6769     { #1 } ,
6770 color .value_required:n = true ,
6771 respect-arraystretch .bool_set:N = \l_@@_respect_arraystretch_bool ,
6772 respect-arraystretch .default:n = true
6773 }

```

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.

```

6774 \cs_new_protected:Npn \@@_Block: { \@@_collect_options:n { \@@_Block_i: } }

6775 \NewExpandableDocumentCommand \@@_Block_i: { m m D < > { } +m }
6776 {

```

If the first mandatory argument of the command (which is the size of the block with the syntax $i-j$) has not be provided by the user, you use 1-1 (that is to say a block of only one cell).

```

6777   \peek_remove_spaces:n
6778   {
6779     \tl_if_blank:nTF { #2 }
6780     { \@@_Block_i 1-1 \q_stop }
6781     {
6782       \int_compare:nNnTF { \char_value_catcode:n { 45 } } = { 13 }
6783       \@@_Block_i_czech \@@_Block_i
6784       #2 \q_stop
6785     }
6786     { #1 } { #3 } { #4 }
6787   }
6788 }

```

With the following construction, we extract the values of i and j in the first mandatory argument of the command.

```

6789 \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`.

```

6790 {
6791   \char_set_catcode_active:N -
6792   \cs_new:Npn \@@_Block_i_czech #1-#2 \q_stop { \@@_Block_ii:nnnnn { #1 } { #2 } }
6793 }

```

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.

```

6794 \cs_new_protected:Npn \@@_Block_ii:nnnnn #1 #2 #3 #4 #5
6795 {

```

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).

```

6796   \bool_lazy_or:nnTF
6797     { \tl_if_blank_p:n { #1 } }
6798     { \str_if_eq_p:nn { #1 } { * } }
6799     { \int_set:Nn \l_tmpa_int { 100 } }
6800     { \int_set:Nn \l_tmpa_int { #1 } }
6801   \bool_lazy_or:nnTF
6802     { \tl_if_blank_p:n { #2 } }
6803     { \str_if_eq_p:nn { #2 } { * } }
6804     { \int_set:Nn \l_tmpb_int { 100 } }
6805     { \int_set:Nn \l_tmpb_int { #2 } }

```

If the block is mono-column.

```

6806   \int_compare:nNnTF \l_tmpb_int = 1
6807     {
6808       \str_if_empty:NTF \l_@@_hpos_cell_str
6809         { \str_set:Nn \l_@@_hpos_block_str c }
6810         { \str_set_eq:NN \l_@@_hpos_block_str \l_@@_hpos_cell_str }
6811     }
6812     { \str_set:Nn \l_@@_hpos_block_str c }

```

The value of \l_@@_hpos_block_str may be modified by the keys of the command \Block that we will analyze now.

```

6813   \keys_set_known:nn { NiceMatrix / Block / FirstPass } { #3 }
6814   \tl_set:Nx \l_tmpa_tl
6815     {
6816       { \int_use:N \c@iRow }
6817       { \int_use:N \c@jCol }
6818       { \int_eval:n { \c@iRow + \l_tmpa_int - 1 } }
6819       { \int_eval:n { \c@jCol + \l_tmpb_int - 1 } }
6820     }

```

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 currying).

```

6821   \bool_if:nTF
6822     {
6823       (
6824         \int_compare_p:nNn { \l_tmpa_int } = 1
6825         ||
6826         \int_compare_p:nNn { \l_tmpb_int } = 1
6827       )
6828     } && ! \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.

```

6829     && ! \l_@@_X_column_bool
6830   }
6831   { \exp_args:Nxx \@@_Block_iv:nnnnn }
6832   { \exp_args:Nxx \@@_Block_v:nnnnn }
6833   { \l_tmpa_int } { \l_tmpb_int } { #3 } { #4 } { #5 }
6834 }

```

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 math mode and before the composition of the block and **#5** is the label (=content) of the block.

```

6835 \cs_new_protected:Npn \@@_Block_iv:nnnnn #1 #2 #3 #4 #5
6836 {
6837   \int_gincr:N \g_@@_block_box_int
6838   \cs_set_protected_nopar:Npn \diagbox ##1 ##2
6839   {
6840     \tl_gput_right:Nx \g_@@_pre_code_after_tl
6841     {
6842       \@@_actually_diagbox:nnnnnn
6843       { \int_use:N \c@iRow }
6844       { \int_use:N \c@jCol }
6845       { \int_eval:n { \c@iRow + #1 - 1 } }
6846       { \int_eval:n { \c@jCol + #2 - 1 } }
6847       { \exp_not:n { ##1 } } { \exp_not:n { ##2 } }
6848     }
6849   }
6850   \box_gclear_new:c
6851   { 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!

```

6852   \hbox_gset:cn
6853   { g_@@_block_box_int \int_use:N \g_@@_block_box_int _ box }
6854   {

```

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}`).

```

6855   \tl_if_empty:NTF \l_@@_color_tl
6856   { \int_compare:nNnT { #2 } = 1 \set@color }
6857   { \@@_color:V \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`.

```

6858   \int_compare:nNnT { #1 } = 1
6859   {
6860     \int_compare:nNnTF \c@iRow = 0
6861     \l_@@_code_for_first_row_tl
6862     {
6863       \int_compare:nNnT \c@iRow = \l_@@_last_row_int
6864       \l_@@_code_for_last_row_tl
6865     }
6866     \g_@@_row_style_tl
6867   }
6868   \bool_if:NF \l_@@_respect_arraystretch_bool
6869   { \cs_set:Npn \arraystretch { 1 } }
6870   \dim_zero:N \extrarowheight

```

#4 is the optional argument of the command `\Block`, provided with the syntax `<...>`.

```

6871   #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.).

```
6872 \@@_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}`.

```
6873 \bool_if:NTF \l_@@_tabular_bool
6874 {
6875   \bool_lazy_all:nTF
6876   {
6877     { \int_compare_p:nNn { #2 } = 1 }
```

Remind that, when the column has not a fixed width, the dimension `\l_@@_col_width_dim` has the conventionnal value of `-1 cm`.

```
6878   { \dim_compare_p:n { \l_@@_col_width_dim >= \c_zero_dim } }
6879   { ! \g_@@_rotate_bool }
6880 }
```

When the block is mono-column in a column with a fixed width (eg `p{3cm}`), we use a `{minipage}`.

```
6881 {
6882   \use:x
6883   {
6884     \exp_not:N \begin { minipage }%
6885     [ \str_lowercase:V { \l_@@_vpos_of_block_str } ]
6886     { \l_@@_col_width_dim }
6887     \str_case:Vn \l_@@_hpos_block_str
6888     { c \centering r \raggedleft l \raggedright }
6889   }
6890   #5
6891   \end { minipage }
6892 }
```

In the other cases, we use a `{tabular}`.

```
6893 {
6894   \use:x
6895   {
6896     \exp_not:N \begin { tabular }%
6897     [ \str_lowercase:V { \l_@@_vpos_of_block_str } ]
6898     { @ { } \l_@@_hpos_block_str @ { } }
6899   }
6900   #5
6901   \end { tabular }
6902 }
6903 }
```

If we are in a mathematical array (`\l_@@_tabular_bool` is false). The composition is always done with an `{array}` (never with a `{minipage}`).

```
6904 {
6905   \c_math_toggle_token
6906   \use:x
6907   {
6908     \exp_not:N \begin { array }%
6909     [ \str_lowercase:V { \l_@@_vpos_of_block_str } ]
6910     { @ { } \l_@@_hpos_block_str @ { } }
6911   }
6912   #5
6913   \end { array }
6914   \c_math_toggle_token
6915 }
6916 }
```

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).

```
6917 \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.

```
6918 \int_compare:nNnT { #2 } = 1
6919 {
6920   \dim_gset:Nn \g_@@_blocks_wd_dim
6921   {
6922     \dim_max:nn
6923     \g_@@_blocks_wd_dim
6924     {
6925       \box_wd:c
6926       { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
6927     }
6928   }
6929 }
```

If we are in a mono-row block, we take into account the height and the depth of that block for the height and the depth of the row.

```
6930 \int_compare:nNnT { #1 } = 1
6931 {
6932   \dim_gset:Nn \g_@@_blocks_ht_dim
6933   {
6934     \dim_max:nn
6935     \g_@@_blocks_ht_dim
6936     {
6937       \box_ht:c
6938       { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
6939     }
6940   }
6941   \dim_gset:Nn \g_@@_blocks_dp_dim
6942   {
6943     \dim_max:nn
6944     \g_@@_blocks_dp_dim
6945     {
6946       \box_dp:c
6947       { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
6948     }
6949   }
6950 }
6951 \seq_gput_right:Nx \g_@@_blocks_seq
6952 {
6953   \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.

```
6954 {
6955   \exp_not:n { #3 } ,
6956   \l_@@_hpos_block_str ,
```

Now, we put a key for the vertical alignment.

```
6957 \bool_if:NT \g_@@_rotate_bool
6958 {
6959   \bool_if:NTF \g_@@_rotate_c_bool
6960   { v-center }
6961   { \int_compare:nNnT \c@iRow = \l_@@_last_row_int T }
6962 }
6963
```

```

6964     }
6965     {
6966         \box_use_drop:c
6967         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
6968     }
6969     }
6970     \bool_set_false:N \g_@@_rotate_c_bool
6971 }

6972 \cs_new:Npn \@@_adjust_hpos_rotate:
6973 {
6974     \bool_if:NT \g_@@_rotate_bool
6975     {
6976         \str_set:Nx \l_@@_hpos_block_str
6977         {
6978             \bool_if:NTF \g_@@_rotate_c_bool
6979             { c }
6980             {
6981                 \str_case:VnF \l_@@_vpos_of_block_str
6982                 { b l B l t r T r }
6983                 { \int_compare:nNnTF \c@iRow = \l_@@_last_row_int r l }
6984             }
6985         }
6986     }
6987 }

```

Despite its name the following command rotates the box of the block *but also does vertical adjustment of the baseline of the block*.

```

6988 \cs_new_protected:Npn \@@_rotate_box_of_block:
6989 {
6990     \box_grotate:cn
6991     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
6992     { 90 }
6993     \int_compare:nNnT \c@iRow = \l_@@_last_row_int
6994     {
6995         \vbox_gset_top:cn
6996         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
6997         {
6998             \skip_vertical:n { 0.8 ex }
6999             \box_use:c
7000             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7001         }
7002     }
7003     \bool_if:NT \g_@@_rotate_c_bool
7004     {
7005         \hbox_gset:cn
7006         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7007         {
7008             \c_math_toggle_token
7009             \vcenter
7010             {
7011                 \box_use:c
7012                 { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7013             }
7014             \c_math_toggle_token
7015         }
7016     }
7017 }

```

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.

```

7018 \cs_new_protected:Npn \@@_Block_v:nnnnn #1 #2 #3 #4 #5
7019 {
7020   \seq_gput_right:Nx \g_@@_blocks_seq
7021   {
7022     \l_tmpa_tl
7023     { \exp_not:n { #3 } }
7024     {
7025       \bool_if:NTF \l_@@_tabular_bool
7026       {
7027         \group_begin:
7028         \bool_if:NF \l_@@_respect_arraystretch_bool
7029         { \cs_set:Npn \exp_not:N \arraystretch { 1 } }
7030         \exp_not:n
7031         {
7032           \dim_zero:N \extrarowheight
7033           #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).

```

7034         % \@@_adjust_hpos_rotate:
7035         \use:x
7036         {
7037           \exp_not:N \begin { tabular } [ \l_@@_vpos_of_block_str ]
7038           { @ { } \l_@@_hpos_block_str @ { } }
7039         }
7040         #5
7041         \end { tabular }
7042       }
7043     \group_end:
7044   }

```

When we are *not* in an environments `{NiceTabular}` (or similar).

```

7045     {
7046       \group_begin:
7047       \bool_if:NF \l_@@_respect_arraystretch_bool
7048       { \cs_set:Npn \exp_not:N \arraystretch { 1 } }
7049       \exp_not:n
7050       {
7051         \dim_zero:N \extrarowheight
7052         #4
7053         % \@@_adjust_hpos_rotate:
7054         \c_math_toggle_token % :n c
7055         \use:x
7056         {
7057           \exp_not:N \begin { array } [ \l_@@_vpos_of_block_str ]
7058           { @ { } \l_@@_hpos_block_str @ { } }
7059         }
7060         #5
7061         \end { array }
7062         \c_math_toggle_token
7063       }
7064     \group_end:
7065   }
7066 }
7067 }
7068 }

```

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).

```

7069 \keys_define:n { NiceMatrix / Block / SecondPass }
7070 {
7071   tikz .code:n =
7072     \IfPackageLoadedTF { tikz }
7073       { \seq_put_right:Nn \l_@@_tikz_seq { { #1 } } }
7074       { \@@_error:n { tikz-key-without-tikz } } ,
7075   tikz .value_required:n = true ,
7076   fill .code:n =
7077     \tl_set_rescan:Nnn
7078     \l_@@_fill_tl
7079     { \char_set_catcode_other:N ! }
7080     { #1 } ,
7081   fill .value_required:n = true ,
7082   opacity .tl_set:N = \l_@@_opacity_tl ,
7083   opacity .value_required:n = true ,
7084   draw .code:n =
7085     \tl_set_rescan:Nnn
7086     \l_@@_draw_tl
7087     { \char_set_catcode_other:N ! }
7088     { #1 } ,
7089   draw .default:n = default ,
7090   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
7091   rounded-corners .default:n = 4 pt ,
7092   color .code:n =
7093     \@@_color:n { #1 }
7094     \tl_set_rescan:Nnn
7095     \l_@@_draw_tl
7096     { \char_set_catcode_other:N ! }
7097     { #1 } ,
7098   borders .clist_set:N = \l_@@_borders_clist ,
7099   borders .value_required:n = true ,
7100   hvlines .meta:n = { vlines , hlines } ,
7101   vlines .bool_set:N = \l_@@_vlines_block_bool ,
7102   vlines .default:n = true ,
7103   hlines .bool_set:N = \l_@@_hlines_block_bool ,
7104   hlines .default:n = true ,
7105   line-width .dim_set:N = \l_@@_line_width_dim ,
7106   line-width .value_required:n = true ,

```

Some keys have not a property `.value_required:n` (or similar) because they are in `FirstPass`.

```

7107   l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7108   r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7109   c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7110   L .code:n = \str_set:Nn \l_@@_hpos_block_str l
7111     \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7112   R .code:n = \str_set:Nn \l_@@_hpos_block_str r
7113     \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7114   C .code:n = \str_set:Nn \l_@@_hpos_block_str c
7115     \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7116   t .code:n = \str_set:Nn \l_@@_vpos_of_block_str t ,
7117   T .code:n = \str_set:Nn \l_@@_vpos_of_block_str T ,
7118   b .code:n = \str_set:Nn \l_@@_vpos_of_block_str b ,
7119   B .code:n = \str_set:Nn \l_@@_vpos_of_block_str B ,
7120   v-center .code:n = \str_set:Nn \l_@@_vpos_of_block_str { c } ,
7121   v-center .value_forbidden:n = true ,
7122   name .tl_set:N = \l_@@_block_name_str ,
7123   name .value_required:n = true ,
7124   name .initial:n = ,
7125   respect-arraystretch .bool_set:N = \l_@@_respect_arraystretch_bool ,
7126   transparent .bool_set:N = \l_@@_transparent_bool ,

```

```

7127 transparent .default:n = true ,
7128 transparent .initial:n = false ,
7129 unknown .code:n = \@@_error:n { Unknown-key-for-Block }
7130 }

```

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.

```

7131 \cs_new_protected:Npn \@@_draw_blocks:
7132 {
7133   \cs_set_eq:NN \ialign \@@_old_ialign:
7134   \seq_map_inline:Nn \g_@@_blocks_seq { \@@_Block_iv:nnnnn ##1 }
7135 }
7136 \cs_new_protected:Npn \@@_Block_iv:nnnnn #1 #2 #3 #4 #5 #6
7137 {

```

The integer `\l_@@_last_row_int` will be the last row of the block and `\l_@@_last_col_int` its last column.

```

7138   \int_zero_new:N \l_@@_last_row_int
7139   \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.

```

7140   \int_compare:nNnTF { #3 } > { 99 }
7141     { \int_set_eq:NN \l_@@_last_row_int \c{iRow }
7142       { \int_set:Nn \l_@@_last_row_int { #3 } }
7143   \int_compare:nNnTF { #4 } > { 99 }
7144     { \int_set_eq:NN \l_@@_last_col_int \c{jCol }
7145       { \int_set:Nn \l_@@_last_col_int { #4 } }
7146   \int_compare:nNnTF \l_@@_last_col_int > \g_@@_col_total_int
7147     {
7148       \int_compare:nTF
7149         { \l_@@_last_col_int <= \g_@@_static_num_of_col_int }
7150         {
7151           \msg_error:nnnn { nicematrix } { Block-too-large~2 } { #1 } { #2 }
7152           \@@_msg_redirect_name:nn { Block-too-large~2 } { none }
7153           \@@_msg_redirect_name:nn { columns-not-used } { none }
7154         }
7155         { \msg_error:nnnn { nicematrix } { Block-too-large~1 } { #1 } { #2 } }
7156       }
7157     {
7158       \int_compare:nNnTF \l_@@_last_row_int > \g_@@_row_total_int
7159         { \msg_error:nnnn { nicematrix } { Block-too-large~1 } { #1 } { #2 } }
7160         { \@@_Block_v:nnnnn { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } }
7161       }
7162     }

```

The following command `\@@_Block_v:nnnnn` 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

```

7163 \cs_new_protected:Npn \@@_Block_v:nnnnn #1 #2 #3 #4 #5 #6
7164 {

```

The group is for the keys.

```

7165   \group_begin:
7166   \int_compare:nNnT { #1 } = { #3 }
7167     { \str_set:Nn \l_@@_vpos_of_block_str { t } }
7168   \keys_set:nn { NiceMatrix / Block / SecondPass } { #5 }

```

```

7169 \bool_if:NT \l_@@_vlines_block_bool
7170 {
7171   \tl_gput_right:Nx \g_nicematrix_code_after_tl
7172   {
7173     \@@_vlines_block:nnn
7174     { \exp_not:n { #5 } }
7175     { #1 - #2 }
7176     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7177   }
7178 }
7179 \bool_if:NT \l_@@_hlines_block_bool
7180 {
7181   \tl_gput_right:Nx \g_nicematrix_code_after_tl
7182   {
7183     \@@_hlines_block:nnn
7184     { \exp_not:n { #5 } }
7185     { #1 - #2 }
7186     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7187   }
7188 }
7189 \bool_if:nF
7190 {
7191   \l_@@_transparent_bool
7192   || ( \l_@@_vlines_block_bool && \l_@@_hlines_block_bool )
7193 }
7194 {

```

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).

```

7195   \seq_gput_left:Nx \g_@@_pos_of_blocks_seq
7196   { { #1 } { #2 } { #3 } { #4 } { \l_@@_block_name_str } }
7197 }

7198 \bool_lazy_and:nnT
7199 { ! (\tl_if_empty_p:N \l_@@_draw_tl) }
7200 { \l_@@_hlines_block_bool || \l_@@_vlines_block_bool }
7201 { \@@_error:n { hlines-with-color } }

7202 \tl_if_empty:NF \l_@@_draw_tl
7203 {
7204   \tl_gput_right:Nx \g_nicematrix_code_after_tl
7205   {
7206     \@@_stroke_block:nnn
7207     { \exp_not:n { #5 } }
7208     { #1 - #2 }
7209     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7210   }
7211   \seq_gput_right:Nn \g_@@_pos_of_stroken_blocks_seq
7212   { { #1 } { #2 } { #3 } { #4 } }
7213 }

7214 \clist_if_empty:NF \l_@@_borders_clist
7215 {
7216   \tl_gput_right:Nx \g_nicematrix_code_after_tl
7217   {
7218     \@@_stroke_borders_block:nnn
7219     { \exp_not:n { #5 } }
7220     { #1 - #2 }
7221     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7222   }
7223 }

```

```

7224 \tl_if_empty:NF \l_@@_fill_tl
7225 {
7226   \tl_if_empty:NF \l_@@_opacity_tl
7227   {
7228     \tl_if_head_eq_meaning:nNTF \l_@@_fill_tl [
7229     {
7230       \tl_set:Nx \l_@@_fill_tl
7231       {
7232         [ opacity = \l_@@_opacity_tl ,
7233         \tl_tail:V \l_@@_fill_tl
7234       }
7235     }
7236     {
7237       \tl_set:Nx \l_@@_fill_tl
7238       { [ opacity = \l_@@_opacity_tl ] { \l_@@_fill_tl } }
7239     }
7240   }
7241   \tl_gput_right:Nx \g_@@_pre_code_before_tl
7242   {
7243     \exp_not:N \roundedrectanglecolor
7244     \exp_args:NV \tl_if_head_eq_meaning:nNTF \l_@@_fill_tl [
7245     { \l_@@_fill_tl }
7246     { { \l_@@_fill_tl } }
7247     { #1 - #2 }
7248     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7249     { \dim_use:N \l_@@_rounded_corners_dim }
7250   }
7251 }
7252 \seq_if_empty:NF \l_@@_tikz_seq
7253 {
7254   \tl_gput_right:Nx \g_nicematrix_code_before_tl
7255   {
7256     \@@_block_tikz:nnnnn
7257     { #1 }
7258     { #2 }
7259     { \int_use:N \l_@@_last_row_int }
7260     { \int_use:N \l_@@_last_col_int }
7261     { \seq_use:Nn \l_@@_tikz_seq { , } }
7262   }
7263 }
7264 \cs_set_protected_nopar:Npn \diagbox ##1 ##2
7265 {
7266   \tl_gput_right:Nx \g_@@_pre_code_after_tl
7267   {
7268     \@@_actually_diagbox:nnnnn
7269     { #1 }
7270     { #2 }
7271     { \int_use:N \l_@@_last_row_int }
7272     { \int_use:N \l_@@_last_col_int }
7273     { \exp_not:n { ##1 } } { \exp_not:n { ##2 } }
7274   }
7275 }
7276 \hbox_set:Nn \l_@@_cell_box { \set@color #6 }
7277 \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.

```

7278 \pgfpicture
7279 \pgfrememberpicturepositiononpagetrue
7280 \pgf@relevantforpicturesizefalse
7281 \@@_qpoint:n { row - #1 }
7282 \dim_set_eq:NN \l_tmpa_dim \pgf@y
7283 \@@_qpoint:n { col - #2 }
7284 \dim_set_eq:NN \l_tmpb_dim \pgf@x
7285 \@@_qpoint:n { row - \int_eval:n { \l_@@_last_row_int + 1 } }
7286 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7287 \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7288 \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.

```

7289 \@@_pgf_rect_node:nnnnn
7290 { \@@_env: - #1 - #2 - block }
7291 \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
7292 \str_if_empty:NF \l_@@_block_name_str
7293 {
7294 \pgfnodealias
7295 { \@@_env: - \l_@@_block_name_str }
7296 { \@@_env: - #1 - #2 - block }
7297 \str_if_empty:NF \l_@@_name_str
7298 {
7299 \pgfnodealias
7300 { \l_@@_name_str - \l_@@_block_name_str }
7301 { \@@_env: - #1 - #2 - block }
7302 }
7303 }

```

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.

```

7304 \bool_if:NF \l_@@_hpos_of_block_cap_bool
7305 {
7306 \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.

```

7307 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
7308 {

```

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.

```

7309 \cs_if_exist:cT
7310 { pgf @ sh @ ns @ \@@_env: - #1 - #2 }

```

```

7311     {
7312     \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
7313     {
7314     \pgfpointanchor { \@@_env: - ##1 - #2 } { west }
7315     \dim_set:Nn \l_tmpb_dim { \dim_min:nn \l_tmpb_dim \pgf@x }
7316     }
7317     }
7318 }

```

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.

```

7319     \dim_compare:nNnT \l_tmpb_dim = \c_max_dim
7320     {
7321     \@@_qpoint:n { col - #2 }
7322     \dim_set_eq:NN \l_tmpb_dim \pgf@x
7323     }
7324 \dim_set:Nn \l_@@_tmpd_dim { - \c_max_dim }
7325 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
7326 {
7327 \cs_if_exist:cT
7328 { \pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
7329 {
7330 \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
7331 {
7332 \pgfpointanchor
7333 { \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
7334 { east }
7335 \dim_set:Nn \l_@@_tmpd_dim { \dim_max:nn \l_@@_tmpd_dim \pgf@x }
7336 }
7337 }
7338 }
7339 \dim_compare:nNnT \l_@@_tmpd_dim = { - \c_max_dim }
7340 {
7341 \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7342 \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
7343 }
7344 \@@_pgf_rect_node:nnnn
7345 { \@@_env: - #1 - #2 - block - short }
7346 \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
7347 }

```

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.

```

7348 \bool_if:NT \l_@@_medium_nodes_bool
7349 {
7350 \@@_pgf_rect_node:nnn
7351 { \@@_env: - #1 - #2 - block - medium }
7352 { \pgfpointanchor { \@@_env: - #1 - #2 - medium } { north~west } }
7353 {
7354 \pgfpointanchor
7355 { \@@_env:
7356 - \int_use:N \l_@@_last_row_int
7357 - \int_use:N \l_@@_last_col_int - medium
7358 }
7359 { south~east }
7360 }
7361 }

```

Now, we will put the label of the block.

```

7362 \bool_lazy_any:nTF
7363 {
7364 { \str_if_eq_p:Vn \l_@@_vpos_of_block_str { c } }
7365 { \str_if_eq_p:Vn \l_@@_vpos_of_block_str { T } }
7366 { \str_if_eq_p:Vn \l_@@_vpos_of_block_str { B } }

```

```

7367     }
7368     {
If we are in the first column, we must put the block as if it was with the key r.
7369     \int_compare:nNnT { #2 } = 0 { \str_set:Nn \l_@@_hpos_block_str r }
If we are in the last column, we must put the block as if it was with the key l.
7370     \bool_if:nT \g_@@_last_col_found_bool
7371     {
7372         \int_compare:nNnT { #2 } = \g_@@_col_total_int
7373         { \str_set:Nn \l_@@_hpos_block_str l }
7374     }
\l_tmpa_tl will contain the anchor of the PGF node which will be used.
7375     \tl_set:Nx \l_tmpa_tl
7376     {
7377         \str_case:Vn \l_@@_vpos_of_block_str
7378         {
7379             c {
7380                 \str_case:Vn \l_@@_hpos_block_str
7381                 {
7382                     c { center }
7383                     l { west }
7384                     r { east }
7385                 }
7386             }
7387         }
7388         T {
7389             \str_case:Vn \l_@@_hpos_block_str
7390             {
7391                 c { north }
7392                 l { north-west }
7393                 r { north-east }
7394             }
7395         }
7396     }
7397     B {
7398         \str_case:Vn \l_@@_hpos_block_str
7399         {
7400             c { south }
7401             l { south-west }
7402             r { south-east }
7403         }
7404     }
7405 }
7406 }
7407 }
7408 \pgftransformshift
7409 {
7410     \pgfpointanchor
7411     {
7412         \@@_env: - #1 - #2 - block
7413         \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
7414     }
7415     { \l_tmpa_tl }
7416 }
7417 \pgfset
7418 {
7419     inner~xsep = \c_zero_dim ,
7420     inner~ysep = \l_@@_block_ysep_dim
7421 }
7422 \pgfnode
7423 { rectangle }

```

```

7424     { \l_tmpa_tl }
7425     { \box_use_drop:N \l_@@_cell_box } { } { }
7426 }

```

End of the case when `\l_@@_vpos_of_block_str` is equal to `c`, `T` or `B`. Now, the other cases.

```

7427 {
7428   \pgfextracty \l_tmpa_dim
7429   {
7430     \@@_qpoint:n
7431     {
7432       row - \str_if_eq:VnTF \l_@@_vpos_of_block_str { b } { #3 } { #1 }
7433       - base
7434     }
7435   }
7436   \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.

```

7437   \pgfpoinanchor
7438   {
7439     \@@_env: - #1 - #2 - block
7440     \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
7441   }
7442   {
7443     \str_case:Vn \l_@@_hpos_block_str
7444     {
7445       c { center }
7446       l { west }
7447       r { east }
7448     }
7449   }

```

We put the label of the block which has been composed in `\l_@@_cell_box`.

```

7450   \pgftransformshift { \pgfpoint \pgf@x \l_tmpa_dim }
7451   \pgfset { inner-sep = \c_zero_dim }
7452   \pgfnode
7453   { rectangle }
7454   {
7455     \str_case:Vn \l_@@_hpos_block_str
7456     {
7457       c { base }
7458       l { base~west }
7459       r { base~east }
7460     }
7461   }
7462   { \box_use_drop:N \l_@@_cell_box } { } { }
7463 }
7464 \endpgfpicture
7465 \group_end:
7466 }

```

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).

```

7467 \cs_new_protected:Npn \@@_stroke_block:nnn #1 #2 #3
7468 {
7469   \group_begin:
7470   \tl_clear:N \l_@@_draw_tl
7471   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
7472   \keys_set_known:nn { NiceMatrix / BlockStroke } { #1 }
7473   \pgfpicture
7474   \pgfrememberpicturepositiononpagetrue
7475   \pgf@relevantforpicturesizefalse
7476   \tl_if_empty:NF \l_@@_draw_tl
7477   {

```

If the user has used the key `color` of the command `\Block` without value, the color fixed by `\arrayrulecolor` is used.

```

7478     \str_if_eq:VnTF \l_@@_draw_tl { default }
7479     { \CT@arc@ }
7480     { \@@_color:V \l_@@_draw_tl }
7481   }
7482   \pgfsetcornersarced
7483   {
7484     \pgfpoint
7485     { \l_@@_rounded_corners_dim }
7486     { \l_@@_rounded_corners_dim }
7487   }
7488   \@@_cut_on_hyphen:w #2 \q_stop
7489   \bool_lazy_and:nnT
7490   { \int_compare_p:n { \l_tmpa_tl <= \c@iRow } }
7491   { \int_compare_p:n { \l_tmpb_tl <= \c@jCol } }
7492   {
7493     \@@_qpoint:n { row - \l_tmpa_tl }
7494     \dim_set_eq:NN \l_tmpb_dim \pgf@y
7495     \@@_qpoint:n { col - \l_tmpb_tl }
7496     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
7497     \@@_cut_on_hyphen:w #3 \q_stop
7498     \int_compare:nNnT \l_tmpa_tl > \c@iRow
7499     { \tl_set:Nx \l_tmpa_tl { \int_use:N \c@iRow } }
7500     \int_compare:nNnT \l_tmpb_tl > \c@jCol
7501     { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@jCol } }
7502     \@@_qpoint:n { row - \int_eval:n { \l_tmpa_tl + 1 } }
7503     \dim_set_eq:NN \l_tmpa_dim \pgf@y
7504     \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
7505     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
7506     \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
7507     \pgfpathrectanglecorners
7508     { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
7509     { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
7510     \dim_compare:nNnTF \l_@@_rounded_corners_dim = \c_zero_dim
7511     { \pgfusepathqstroke }
7512     { \pgfusepath { stroke } }
7513   }
7514   \endpgfpicture
7515   \group_end:
7516 }

```

Here is the set of keys for the command `\@@_stroke_block:nnn`.

```

7517 \keys_define:nn { NiceMatrix / BlockStroke }
7518 {
7519   color .tl_set:N = \l_@@_draw_tl ,
7520   draw .code:n =
7521     \exp_args:Nx \tl_if_empty:nF { #1 } { \tl_set:Nn \l_@@_draw_tl { #1 } } ,
7522   draw .default:n = default ,
7523   line-width .dim_set:N = \l_@@_line_width_dim ,
7524   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
7525   rounded-corners .default:n = 4 pt
7526 }

```

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).

```

7527 \cs_new_protected:Npn \@@_vlines_block:nnn #1 #2 #3
7528 {
7529   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
7530   \keys_set_known:nn { NiceMatrix / BlockBorders } { #1 }
7531   \@@_cut_on_hyphen:w #2 \q_stop
7532   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl

```

```

7533 \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
7534 \@@_cut_on_hyphen:w #3 \q_stop
7535 \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
7536 \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
7537 \int_step_inline:nnn \l_@@_tmpd_tl \l_tmpb_tl
7538 {
7539   \use:x
7540   {
7541     \@@_vline:n
7542     {
7543       position = ##1 ,
7544       start = \l_@@_tmpc_tl ,
7545       end = \int_eval:n { \l_tmpa_tl - 1 } ,
7546       total-width = \dim_use:N \l_@@_line_width_dim % added 2022-08-06
7547     }
7548   }
7549 }
7550 }
7551 \cs_new_protected:Npn \@@_hlines_block:nnn #1 #2 #3
7552 {
7553   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
7554   \keys_set_known:nn { NiceMatrix / BlockBorders } { #1 }
7555   \@@_cut_on_hyphen:w #2 \q_stop
7556   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
7557   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
7558   \@@_cut_on_hyphen:w #3 \q_stop
7559   \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
7560   \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
7561   \int_step_inline:nnn \l_@@_tmpc_tl \l_tmpa_tl
7562   {
7563     \use:x
7564     {
7565       \@@_hline:n
7566       {
7567         position = ##1 ,
7568         start = \l_@@_tmpd_tl ,
7569         end = \int_eval:n { \l_tmpb_tl - 1 } ,
7570         total-width = \dim_use:N \l_@@_line_width_dim
7571       }
7572     }
7573   }
7574 }

```

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).

```

7575 \cs_new_protected:Npn \@@_stroke_borders_block:nnn #1 #2 #3
7576 {
7577   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
7578   \keys_set_known:nn { NiceMatrix / BlockBorders } { #1 }
7579   \dim_compare:nNnTF \l_@@_rounded_corners_dim > \c_zero_dim
7580   { \@@_error:n { borders~forbidden } }
7581   {
7582     \tl_clear_new:N \l_@@_borders_tikz_tl
7583     \keys_set:nV
7584     { NiceMatrix / OnlyForTikzInBorders }
7585     \l_@@_borders_clist
7586     \@@_cut_on_hyphen:w #2 \q_stop
7587     \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
7588     \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
7589     \@@_cut_on_hyphen:w #3 \q_stop
7590     \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
7591     \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }

```

```

7592     \@@_stroke_borders_block_i:
7593   }
7594 }

7595 \hook_ggput_code:nnn { begindocument } { . }
7596 {
7597   \cs_new_protected:Npx \@@_stroke_borders_block_i:
7598   {
7599     \c_@@_pgfortikzpicture_tl
7600     \@@_stroke_borders_block_ii:
7601     \c_@@_endpgfortikzpicture_tl
7602   }
7603 }

7604 \cs_new_protected:Npn \@@_stroke_borders_block_ii:
7605 {
7606   \pgfrememberpicturepositiononpagetrue
7607   \pgf@relevantforpicturesizefalse
7608   \CT@arc@
7609   \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
7610   \clist_if_in:NnT \l_@@_borders_clist { right }
7611   { \@@_stroke_vertical:n \l_tmpb_tl }
7612   \clist_if_in:NnT \l_@@_borders_clist { left }
7613   { \@@_stroke_vertical:n \l_@@_tmpd_tl }
7614   \clist_if_in:NnT \l_@@_borders_clist { bottom }
7615   { \@@_stroke_horizontal:n \l_tmpa_tl }
7616   \clist_if_in:NnT \l_@@_borders_clist { top }
7617   { \@@_stroke_horizontal:n \l_@@_tmpc_tl }
7618 }

7619 \keys_define:nn { NiceMatrix / OnlyForTikzInBorders }
7620 {
7621   tikz .code:n =
7622     \cs_if_exist:NTF \tikzpicture
7623     { \tl_set:Nn \l_@@_borders_tikz_tl { #1 } }
7624     { \@@_error:n { tikz-in~borders-without-tikz } } ,
7625   tikz .value_required:n = true ,
7626   top .code:n = ,
7627   bottom .code:n = ,
7628   left .code:n = ,
7629   right .code:n = ,
7630   unknown .code:n = \@@_error:n { bad~border }
7631 }

```

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).

```

7632 \cs_new_protected:Npn \@@_stroke_vertical:n #1
7633 {
7634   \@@_qpoint:n \l_@@_tmpc_tl
7635   \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
7636   \@@_qpoint:n \l_tmpa_tl
7637   \dim_set:Nn \l_@@_tmpc_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
7638   \@@_qpoint:n { #1 }
7639   \tl_if_empty:NTF \l_@@_borders_tikz_tl
7640   {
7641     \pgfpathmoveto { \pgfpoint \pgf@x \l_tmpb_dim }
7642     \pgfpathlineto { \pgfpoint \pgf@x \l_@@_tmpc_dim }
7643     \pgfusepathqstroke
7644   }
7645   {
7646     \use:x { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
7647     ( \pgf@x , \l_tmpb_dim ) -- ( \pgf@x , \l_@@_tmpc_dim ) ;
7648   }
7649 }

```

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).

```

7650 \cs_new_protected:Npn \@@_stroke_horizontal:n #1
7651   {
7652     \@@_qpoint:n \l_@@_tmpd_tl
7653     \clist_if_in:NnTF \l_@@_borders_clist { left }
7654       { \dim_set:Nn \l_tmpa_dim { \pgf@x - 0.5 \l_@@_line_width_dim } }
7655       { \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \l_@@_line_width_dim } }
7656     \@@_qpoint:n \l_tmpb_tl
7657     \dim_set:Nn \l_tmpb_dim { \pgf@x + 0.5 \l_@@_line_width_dim }
7658     \@@_qpoint:n { #1 }
7659     \tl_if_empty:NTF \l_@@_borders_tikz_tl
7660       {
7661         \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }
7662         \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
7663         \pgfusepathqstroke
7664       }
7665       {
7666         \use:x { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
7667           ( \l_tmpa_dim , \pgf@y ) -- ( \l_tmpb_dim , \pgf@y ) ;
7668       }
7669   }

```

Here is the set of keys for the command \@@_stroke_borders_block:nnn.

```

7670 \keys_define:nn { NiceMatrix / BlockBorders }
7671   {
7672     borders .clist_set:N = \l_@@_borders_clist ,
7673     rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
7674     rounded-corners .default:n = 4 pt ,
7675     line-width .dim_set:N = \l_@@_line_width_dim ,
7676   }

```

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.

```

7677 \cs_new_protected:Npn \@@_block_tikz:nnnnn #1 #2 #3 #4 #5
7678   {
7679     \begin { tikzpicture }
7680     \@@_clip_with_rounded_corners:
7681     \clist_map_inline:nn { #5 }
7682       {
7683         \path [ ##1 ]
7684           ( #1 -| #2 )
7685           rectangle
7686           ( \int_eval:n { #3 + 1 } -| \int_eval:n { #4 + 1 } ) ;
7687       }
7688     \end { tikzpicture }
7689   }

```

28 How to draw the dotted lines transparently

```

7690 \cs_set_protected:Npn \@@_renew_matrix:
7691   {
7692     \RenewDocumentEnvironment { pmatrix } { } {
7693       { \pNiceMatrix }
7694       { \endpNiceMatrix }
7695     }
7696     \RenewDocumentEnvironment { vmatrix } { } {
7697       { \vNiceMatrix }
7698       { \endvNiceMatrix }

```



```

7698 \RenewDocumentEnvironment { Vmatrix } { }
7699   { \VNiceMatrix }
7700   { \endVNiceMatrix }
7701 \RenewDocumentEnvironment { bmatrix } { }
7702   { \bNiceMatrix }
7703   { \endbNiceMatrix }
7704 \RenewDocumentEnvironment { Bmatrix } { }
7705   { \BNiceMatrix }
7706   { \endBNiceMatrix }
7707 }

```

29 Automatic arrays

We will extract some keys and pass the other keys to the environment `{NiceArrayWithDelims}`.

```

7708 \keys_define:nn { NiceMatrix / Auto }
7709   {
7710     columns-type .code:n = \@@_set_preamble:Nn \l_@@_columns_type_tl { #1 } ,
7711     columns-type .value_required:n = true ,
7712     l .meta:n = { columns-type = l } ,
7713     r .meta:n = { columns-type = r } ,
7714     c .meta:n = { columns-type = c } ,
7715     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
7716     delimiters / color .value_required:n = true ,
7717     delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
7718     delimiters / max-width .default:n = true ,
7719     delimiters .code:n = \keys_set:nn { NiceMatrix / delimiters } { #1 } ,
7720     delimiters .value_required:n = true ,
7721     rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
7722     rounded-corners .default:n = 4 pt
7723   }
7724 \NewDocumentCommand \AutoNiceMatrixWithDelims
7725   { m m O { } } > { \SplitArgument { 1 } { - } } m O { } m ! O { } }
7726   { \@@_auto_nice_matrix:nnnnn { #1 } { #2 } #4 { #6 } { #3 , #5 , #7 } }
7727 \cs_new_protected:Npn \@@_auto_nice_matrix:nnnnn #1 #2 #3 #4 #5 #6
7728   {

```

The group is for the protection of the keys.

```

7729   \group_begin:
7730   \keys_set_known:nnN { NiceMatrix / Auto } { #6 } \l_tmpa_tl

```

We nullify the command `\@@_transform_preamble_i:` because we will provide a preamble which is yet transformed (by using `\l_@@_columns_type_tl` which is yet `nicematrix-ready`).

```

7731   \bool_set_false:N \l_@@_preamble_bool
7732   \use:x
7733   {
7734     \exp_not:N \begin { NiceArrayWithDelims } { #1 } { #2 }
7735     { * { #4 } { \exp_not:V \l_@@_columns_type_tl } }
7736     [ \exp_not:V \l_tmpa_tl ]
7737   }
7738   \int_compare:nNnT \l_@@_first_row_int = 0
7739   {
7740     \int_compare:nNnT \l_@@_first_col_int = 0 { & }
7741     \prg_replicate:nn { #4 - 1 } { & }
7742     \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
7743   }
7744   \prg_replicate:nn { #3 }
7745   {
7746     \int_compare:nNnT \l_@@_first_col_int = 0 { & }

```

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`).

```

7747     \prg_replicate:nN { #4 - 1 } { { } #5 & } #5
7748     \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
7749   }
7750   \int_compare:nNnT \l_@@_last_row_int > { -2 }
7751   {
7752     \int_compare:nNnT \l_@@_first_col_int = 0 { & }
7753     \prg_replicate:nN { #4 - 1 } { & }
7754     \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
7755   }
7756   \end { NiceArrayWithDelims }
7757   \group_end:
7758 }

7759 \cs_set_protected:Npn \@@_define_com:nnn #1 #2 #3
7760 {
7761   \cs_set_protected:cpn { #1 AutoNiceMatrix }
7762   {
7763     \bool_gset_true:N \g_@@_delims_bool
7764     \str_gset:Nx \g_@@_name_env_str { #1 AutoNiceMatrix }
7765     \AutoNiceMatrixWithDelims { #2 } { #3 }
7766   }
7767 }

7768 \@@_define_com:nnn p ( )
7769 \@@_define_com:nnn b [ ]
7770 \@@_define_com:nnn v | |
7771 \@@_define_com:nnn V \ | \ |
7772 \@@_define_com:nnn B \{ \}

```

We define also a command `\AutoNiceMatrix` similar to the environment `{NiceMatrix}`.

```

7773 \NewDocumentCommand \AutoNiceMatrix { 0 { } m 0 { } m ! 0 { } }
7774 {
7775   \group_begin:
7776   \bool_gset_false:N \g_@@_delims_bool
7777   \AutoNiceMatrixWithDelims . . { #2 } { #4 } [ #1 , #3 , #5 ]
7778   \group_end:
7779 }

```

30 The redefinition of the command `\dotfill`

```

7780 \cs_set_eq:NN \@@_old_dotfill \dotfill
7781 \cs_new_protected:Npn \@@_dotfill:
7782 {

```

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}`).

```

7783   \@@_old_dotfill
7784   \tl_gput_right:Nn \g_@@_cell_after_hook_tl \@@_dotfill_i:
7785 }

```

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`.

```

7786 \cs_new_protected:Npn \@@_dotfill_i:
7787 { \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.

```

7788 \cs_new_protected:Npn \@@_diagbox:nn #1 #2
7789   {
7790     \tl_gput_right:Nx \g_@@_pre_code_after_tl
7791     {
7792       \@@_actually_diagbox:nnnnnn
7793       { \int_use:N \c@iRow }
7794       { \int_use:N \c@jCol }
7795       { \int_use:N \c@iRow }
7796       { \int_use:N \c@jCol }
7797       { \exp_not:n { #1 } }
7798       { \exp_not:n { #2 } }
7799     }

```

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`.

```

7800   \seq_gput_right:Nx \g_@@_pos_of_blocks_seq
7801   {
7802     { \int_use:N \c@iRow }
7803     { \int_use:N \c@jCol }
7804     { \int_use:N \c@iRow }
7805     { \int_use:N \c@jCol }

```

The last argument is for the name of the block.

```

7806     { }
7807   }
7808 }

```

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.

```

7809 \cs_new_protected:Npn \@@_actually_diagbox:nnnnnn #1 #2 #3 #4 #5 #6
7810   {
7811     \pgfpicture
7812     \pgf@relevantforpicturesizefalse
7813     \pgfrememberpicturepositiononpagetrue
7814     \@@_qpoint:n { row - #1 }
7815     \dim_set_eq:NN \l_tmpa_dim \pgf@y
7816     \@@_qpoint:n { col - #2 }
7817     \dim_set_eq:NN \l_tmpb_dim \pgf@x
7818     \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
7819     \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
7820     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7821     \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
7822     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
7823     \pgfpathlineto { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
7824     {

```

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.

```

7825     \CT@arc@
7826     \pgfsetroundcap
7827     \pgfusepathqstroke
7828   }
7829   \pgfset { inner~sep = 1 pt }
7830   \pgfscope
7831   \pgftransformshift { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
7832   \pgfnode { rectangle } { south-west }

```

```

7833     {
7834     \begin { minipage } { 20 cm }
7835     \@@_math_toggle_token: #5 \@@_math_toggle_token:
7836     \end { minipage }
7837     }
7838     { }
7839     { }
7840 \endpgfscope
7841 \pgftransformshift { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
7842 \pgfnode { rectangle } { north-east }
7843     {
7844     \begin { minipage } { 20 cm }
7845     \raggedleft
7846     \@@_math_toggle_token: #6 \@@_math_toggle_token:
7847     \end { minipage }
7848     }
7849     { }
7850     { }
7851 \endpgfpicture
7852 }

```

32 The keyword `\CodeAfter`

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.

```

7853 \keys_define:nn { NiceMatrix }
7854   {
7855     CodeAfter / rules .inherit:n = NiceMatrix / rules ,
7856     CodeAfter / sub-matrix .inherit:n = NiceMatrix / sub-matrix
7857   }
7858 \keys_define:nn { NiceMatrix / CodeAfter }
7859   {
7860     sub-matrix .code:n = \keys_set:nn { NiceMatrix / sub-matrix } { #1 } ,
7861     sub-matrix .value_required:n = true ,
7862     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
7863     delimiters / color .value_required:n = true ,
7864     rules .code:n = \keys_set:nn { NiceMatrix / rules } { #1 } ,
7865     rules .value_required:n = true ,
7866     xdots .code:n = \keys_set:nn { NiceMatrix / xdots } { #1 } ,
7867     unknown .code:n = \@@_error:n { Unknown-key-for-CodeAfter }
7868   }

```

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. 80.

In the environments of `nicematrix`, `\CodeAfter` will be linked to `\@@_CodeAfter:`. That macro must *not* be protected since it begins with `\omit`.

```

7869 \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 `\`.

```

7870 \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`.

```

7871 \cs_new_protected:Npn \@@_CodeAfter_ii:n #1 \end
7872   {

```

```

7873 \tl_gput_right:Nn \g_nicematrix_code_after_tl { #1 }
7874 \@@_CodeAfter_iv:n
7875 }

```

We catch the argument of the command `\end` (in `#1`).

```

7876 \cs_new_protected:Npn \@@_CodeAfter_iv:n #1
7877 {

```

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.

```

7878 \str_if_eq:eeTF \currenenvir { #1 }
7879 { \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`.

```

7880 {
7881 \tl_gput_right:Nn \g_nicematrix_code_after_tl { \end { #1 } }
7882 \@@_CodeAfter_ii:n
7883 }
7884 }

```

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.

```

7885 \cs_new_protected:Npn \@@_delimiter:nnn #1 #2 #3
7886 {
7887 \pgfpicture
7888 \pgfrememberpicturepositiononpagetrue
7889 \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.

```

7890 \@@_qpoint:n { row - 1 }
7891 \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
7892 \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
7893 \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).

```

7894 \bool_if:nTF { #3 }
7895 { \dim_set_eq:NN \l_tmpa_dim \c_max_dim }
7896 { \dim_set:Nn \l_tmpa_dim { - \c_max_dim } }
7897 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
7898 {
7899 \cs_if_exist:cT
7900 { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
7901 {
7902 \pgfpointanchor
7903 { \@@_env: - ##1 - #2 }
7904 { \bool_if:nTF { #3 } { west } { east } }

```

```

7905         \dim_set:Nn \l_tmpa_dim
7906         { \bool_if:nTF { #3 } \dim_min:nn \dim_max:nn \l_tmpa_dim \pgf{x }
7907     }
7908 }

```

Now we can put the delimiter with a node of PGF.

```

7909 \pgfset { inner~sep = \c_zero_dim }
7910 \dim_zero:N \nulldelimiterspace
7911 \pgftransformshift
7912 {
7913     \pgfpoint
7914     { \l_tmpa_dim }
7915     { ( \l_@@_y_initial_dim + \l_@@_y_final_dim + \arrayrulewidth ) / 2 }
7916 }
7917 \pgfnode
7918 { rectangle }
7919 { \bool_if:nTF { #3 } { east } { west } }
7920 {

```

Here is the content of the PGF node, that is to say the delimiter, constructed with its right size.

```

7921     \nullfont
7922     \c_math_toggle_token
7923     \@@_color:V \l_@@_delimiters_color_tl
7924     \bool_if:nTF { #3 } { \left #1 } { \left . }
7925     \vcenter
7926     {
7927         \nullfont
7928         \hrule \@height
7929             \dim_eval:n { \l_@@_y_initial_dim - \l_@@_y_final_dim }
7930             \@depth \c_zero_dim
7931             \@width \c_zero_dim
7932     }
7933     \bool_if:nTF { #3 } { \right . } { \right #1 }
7934     \c_math_toggle_token
7935 }
7936 { }
7937 { }
7938 \endpgfpicture
7939 }

```

34 The command `\SubMatrix`

```

7940 \keys_define:nn { NiceMatrix / sub-matrix }
7941 {
7942     extra-height .dim_set:N = \l_@@_submatrix_extra_height_dim ,
7943     extra-height .value_required:n = true ,
7944     left-xshift .dim_set:N = \l_@@_submatrix_left_xshift_dim ,
7945     left-xshift .value_required:n = true ,
7946     right-xshift .dim_set:N = \l_@@_submatrix_right_xshift_dim ,
7947     right-xshift .value_required:n = true ,
7948     xshift .meta:n = { left-xshift = #1, right-xshift = #1 } ,
7949     xshift .value_required:n = true ,
7950     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
7951     delimiters / color .value_required:n = true ,
7952     slim .bool_set:N = \l_@@_submatrix_slim_bool ,
7953     slim .default:n = true ,
7954     hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
7955     hlines .default:n = all ,
7956     vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
7957     vlines .default:n = all ,
7958     hvlines .meta:n = { hlines, vlines } ,

```

```

7959   hvlines .value_forbidden:n = true ,
7960   }
7961 \keys_define:nn { NiceMatrix }
7962   {
7963     SubMatrix .inherit:n = NiceMatrix / sub-matrix ,
7964     CodeAfter / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
7965     NiceMatrix / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
7966     NiceArray / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
7967     pNiceArray / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
7968     NiceMatrixOptions / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
7969   }

```

The following keys set is for the command `\SubMatrix` itself (not the tuning of `\SubMatrix` that can be done elsewhere).

```

7970 \keys_define:nn { NiceMatrix / SubMatrix }
7971   {
7972     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
7973     delimiters / color .value_required:n = true ,
7974     hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
7975     hlines .default:n = all ,
7976     vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
7977     vlines .default:n = all ,
7978     hvlines .meta:n = { hlines, vlines } ,
7979     hvlines .value_forbidden:n = true ,
7980     name .code:n =
7981       \tl_if_empty:nTF { #1 }
7982         { \@@_error:n { Invalid-name } }
7983         {
7984           \regex_match:nnTF { \A[A-Za-z][A-Za-z0-9]*\Z } { #1 }
7985           {
7986             \seq_if_in:NnTF \g_@@_submatrix_names_seq { #1 }
7987               { \@@_error:nn { Duplicate-name-for-SubMatrix } { #1 } }
7988               {
7989                 \str_set:Nn \l_@@_submatrix_name_str { #1 }
7990                 \seq_gput_right:Nn \g_@@_submatrix_names_seq { #1 }
7991               }
7992             }
7993           { \@@_error:n { Invalid-name } }
7994         } ,
7995     name .value_required:n = true ,
7996     rules .code:n = \keys_set:nn { NiceMatrix / rules } { #1 } ,
7997     rules .value_required:n = true ,
7998     code .tl_set:N = \l_@@_code_tl ,
7999     code .value_required:n = true ,
8000     unknown .code:n = \@@_error:n { Unknown-key-for-SubMatrix }
8001   }

8002 \NewDocumentCommand \@@_SubMatrix_in_code_before { m m m m ! 0 { } }
8003   {
8004     \peek_remove_spaces:n
8005     {
8006       \tl_gput_right:Nx \g_@@_pre_code_after_tl
8007       {
8008         \SubMatrix { #1 } { #2 } { #3 } { #4 }
8009         [
8010           delimiters / color = \l_@@_delimiters_color_tl ,
8011           hlines = \l_@@_submatrix_hlines_clist ,
8012           vlines = \l_@@_submatrix_vlines_clist ,
8013           extra-height = \dim_use:N \l_@@_submatrix_extra_height_dim ,
8014           left-xshift = \dim_use:N \l_@@_submatrix_left_xshift_dim ,
8015           right-xshift = \dim_use:N \l_@@_submatrix_right_xshift_dim ,
8016           slim = \bool_to_str:N \l_@@_submatrix_slim_bool ,
8017           #5

```

```

8018         ]
8019     }
8020     \@@_SubMatrix_in_code_before_i { #2 } { #3 }
8021 }
8022 }
8023 \NewDocumentCommand \@@_SubMatrix_in_code_before_i
8024 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
8025 { \@@_SubMatrix_in_code_before_i:nnnn #1 #2 }
8026 \cs_new_protected:Npn \@@_SubMatrix_in_code_before_i:nnnn #1 #2 #3 #4
8027 {
8028     \seq_gput_right:Nx \g_@@_submatrix_seq
8029     {

```

We use `\str_if_eq:nnTF` because it is fully expandable.

```

8030     { \str_if_eq:nnTF { #1 } { last } { \int_use:N \c@iRow } { #1 } }
8031     { \str_if_eq:nnTF { #2 } { last } { \int_use:N \c@jCol } { #2 } }
8032     { \str_if_eq:nnTF { #3 } { last } { \int_use:N \c@iRow } { #3 } }
8033     { \str_if_eq:nnTF { #4 } { last } { \int_use:N \c@jCol } { #4 } }
8034 }
8035 }

```

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`.

```

8036 \hook_gput_code:nnn { begindocument } { . }
8037 {
8038     \tl_set:Nn \l_@@_argspec_tl { m m m m 0 { } E { _ ^ } { { } { } } }
8039     \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
8040     \exp_args:NNV \NewDocumentCommand \@@_SubMatrix \l_@@_argspec_tl
8041     {
8042         \peek_remove_spaces:n
8043         {
8044             \@@_sub_matrix:nnnnnnn
8045             { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } { #7 }
8046         }
8047     }
8048 }

```

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).

```

8049 \NewDocumentCommand \@@_compute_i_j:nn
8050 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
8051 { \@@_compute_i_j:nnnn #1 #2 }

```



```

8052 \cs_new_protected:Npn \@@_compute_i_j:nnnn #1 #2 #3 #4
8053 {
8054   \tl_set:Nn \l_@@_first_i_tl { #1 }
8055   \tl_set:Nn \l_@@_first_j_tl { #2 }
8056   \tl_set:Nn \l_@@_last_i_tl { #3 }
8057   \tl_set:Nn \l_@@_last_j_tl { #4 }
8058   \tl_if_eq:NnT \l_@@_first_i_tl { last }
8059     { \tl_set:NV \l_@@_first_i_tl \c@iRow }
8060   \tl_if_eq:NnT \l_@@_first_j_tl { last }
8061     { \tl_set:NV \l_@@_first_j_tl \c@jCol }
8062   \tl_if_eq:NnT \l_@@_last_i_tl { last }
8063     { \tl_set:NV \l_@@_last_i_tl \c@iRow }
8064   \tl_if_eq:NnT \l_@@_last_j_tl { last }
8065     { \tl_set:NV \l_@@_last_j_tl \c@jCol }
8066 }

8067 \cs_new_protected:Npn \@@_sub_matrix:nnnnnn #1 #2 #3 #4 #5 #6 #7
8068 {
8069   \group_begin:

```

The four following token lists correspond to the position of the \SubMatrix.

```

8070 \@@_compute_i_j:nn { #2 } { #3 }
8071 % added 6.19b
8072 \int_compare:nNnT \l_@@_first_i_tl = \l_@@_last_i_tl
8073   { \cs_set:Npn \arraystretch { 1 } }
8074 \bool_lazy_or:nnTF
8075   { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
8076   { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
8077   { \@@_error:nn { Construct-too~large } { \SubMatrix } }
8078 {
8079   \str_clear_new:N \l_@@_submatrix_name_str
8080   \keys_set:nn { NiceMatrix / SubMatrix } { #5 }
8081   \pgfpicture
8082   \pgfrememberpicturerepositiononpagetrue
8083   \pgf@relevantforpicturesizefalse
8084   \pgfset { inner-sep = \c_zero_dim }
8085   \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
8086   \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }

```

The last value of \int_step_inline:nnn is provided by curification.

```

8087 \bool_if:NTF \l_@@_submatrix_slim_bool
8088   { \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl }
8089   { \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int }
8090   {
8091     \cs_if_exist:cT
8092       { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
8093       {
8094         \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
8095         \dim_set:Nn \l_@@_x_initial_dim
8096           { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
8097       }
8098     \cs_if_exist:cT
8099       { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
8100       {
8101         \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
8102         \dim_set:Nn \l_@@_x_final_dim
8103           { \dim_max:nn \l_@@_x_final_dim \pgf@x }
8104       }
8105     }
8106   \dim_compare:nNnTF \l_@@_x_initial_dim = \c_max_dim
8107     { \@@_error:nn { Impossible-delimiter } { left } }
8108     {
8109       \dim_compare:nNnTF \l_@@_x_final_dim = { - \c_max_dim }
8110         { \@@_error:nn { Impossible~delimiter } { right } }
8111         { \@@_sub_matrix_i:nnnn { #1 } { #4 } { #6 } { #7 } }

```

```

8112     }
8113     \endpgfpicture
8114 }
8115 \group_end:
8116 }

```

#1 is the left delimiter, #2 is the right one, #3 is the subscript and #4 is the superscript.

```

8117 \cs_new_protected:Npn \@@_sub_matrix_i:nnnn #1 #2 #3 #4
8118 {
8119   \@@_qpoint:n { row - \l_@@_first_i_tl - base }
8120   \dim_set:Nn \l_@@_y_initial_dim
8121   {
8122     \fp_to_dim:n
8123     {
8124       \pgf@y
8125       + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
8126     }
8127   } % modified 6.13c
8128   \@@_qpoint:n { row - \l_@@_last_i_tl - base }
8129   \dim_set:Nn \l_@@_y_final_dim
8130   { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
8131   % modified 6.13c
8132   \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
8133   {
8134     \cs_if_exist:cT
8135     { pgf @ sh @ ns @ \@@_env: - \l_@@_first_i_tl - ##1 }
8136     {
8137       \pgfpointanchor { \@@_env: - \l_@@_first_i_tl - ##1 } { north }
8138       \dim_set:Nn \l_@@_y_initial_dim
8139       { \dim_max:nn \l_@@_y_initial_dim \pgf@y }
8140     }
8141     \cs_if_exist:cT
8142     { pgf @ sh @ ns @ \@@_env: - \l_@@_last_i_tl - ##1 }
8143     {
8144       \pgfpointanchor { \@@_env: - \l_@@_last_i_tl - ##1 } { south }
8145       \dim_set:Nn \l_@@_y_final_dim
8146       { \dim_min:nn \l_@@_y_final_dim \pgf@y }
8147     }
8148   }
8149   \dim_set:Nn \l_tmpa_dim
8150   {
8151     \l_@@_y_initial_dim - \l_@@_y_final_dim +
8152     \l_@@_submatrix_extra_height_dim - \arrayrulewidth
8153   }
8154   \dim_zero:N \nulldelimiterspace

```

We will draw the rules in the `\SubMatrix`.

```

8155   \group_begin:
8156   \pgfsetlinewidth { 1.1 \arrayrulewidth }
8157   \@@_set_CT@arc@:V \l_@@_rules_color_tl
8158   \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`.

```

8159   \seq_map_inline:Nn \g_@@_cols_vlism_seq
8160   {
8161     \int_compare:nNnT \l_@@_first_j_tl < { ##1 }
8162     {
8163       \int_compare:nNnT
8164       { ##1 } < { \int_eval:n { \l_@@_last_j_tl + 1 } }
8165       {

```

First, we extract the value of the abscissa of the rule we have to draw.

```

8166         \l_@@_qpoint:n { col - ##1 }
8167         \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
8168         \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
8169         \pgfusepathqstroke
8170     }
8171 }
8172 }

```

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.

```

8173 \tl_if_eq:NnTF \l_@@_submatrix_vlines_clist { all }
8174 { \int_step_inline:nn { \l_@@_last_j_tl - \l_@@_first_j_tl } }
8175 { \clist_map_inline:Nn \l_@@_submatrix_vlines_clist }
8176 {
8177     \bool_lazy_and:nnTF
8178     { \int_compare_p:nNn { ##1 } > 0 }
8179     {
8180         \int_compare_p:nNn
8181         { ##1 } < { \l_@@_last_j_tl - \l_@@_first_j_tl + 1 } }
8182     {
8183         \l_@@_qpoint:n { col - \int_eval:n { ##1 + \l_@@_first_j_tl } }
8184         \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
8185         \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
8186         \pgfusepathqstroke
8187     }
8188     { \@@_error:nnn { Wrong-line-in-SubMatrix } { vertical } { ##1 } }
8189 }

```

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.

```

8190 \tl_if_eq:NnTF \l_@@_submatrix_hlines_clist { all }
8191 { \int_step_inline:nn { \l_@@_last_i_tl - \l_@@_first_i_tl } }
8192 { \clist_map_inline:Nn \l_@@_submatrix_hlines_clist }
8193 {
8194     \bool_lazy_and:nnTF
8195     { \int_compare_p:nNn { ##1 } > 0 }
8196     {
8197         \int_compare_p:nNn
8198         { ##1 } < { \l_@@_last_i_tl - \l_@@_first_i_tl + 1 } }
8199     {
8200         \l_@@_qpoint:n { row - \int_eval:n { ##1 + \l_@@_first_i_tl } }

```

We use a group to protect `\l_tmpa_dim` and `\l_tmpb_dim`.

```

8201     \group_begin:

```

We compute in `\l_tmpa_dim` the x -value of the left end of the rule.

```

8202     \dim_set:Nn \l_tmpa_dim
8203     { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
8204     \str_case:nn { #1 }
8205     {
8206         ( { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
8207         [ { \dim_sub:Nn \l_tmpa_dim { 0.2 mm } }
8208         ] { { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } } }
8209     }
8210     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }

```

We compute in `\l_tmpb_dim` the x -value of the right end of the rule.

```

8211     \dim_set:Nn \l_tmpb_dim
8212     { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
8213     \str_case:nn { #2 }
8214     {
8215         ) { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
8216         ] { \dim_add:Nn \l_tmpb_dim { 0.2 mm } }

```

```

8217         \} { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
8218     }
8219     \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
8220     \pgfusepathqstroke
8221     \group_end:
8222 }
8223 { \@@_error:nnn { Wrong-line-in-SubMatrix } { horizontal } { ##1 } }
8224 }

```

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).

```

8225 \str_if_empty:NF \l_@@_submatrix_name_str
8226 {
8227     \@@_pgf_rect_node:nnnnn \l_@@_submatrix_name_str
8228     \l_@@_x_initial_dim \l_@@_y_initial_dim
8229     \l_@@_x_final_dim \l_@@_y_final_dim
8230 }
8231 \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`.

```

8232 \begin { pgfscope }
8233 \pgftransformshift
8234 {
8235     \pgfpoint
8236     { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
8237     { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
8238 }
8239 \str_if_empty:NTF \l_@@_submatrix_name_str
8240 { \@@_node_left:nn #1 { } }
8241 { \@@_node_left:nn #1 { \@@_env: - \l_@@_submatrix_name_str - left } }
8242 \end { pgfscope }

```

Now, we deal with the right delimiter.

```

8243 \pgftransformshift
8244 {
8245     \pgfpoint
8246     { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
8247     { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
8248 }
8249 \str_if_empty:NTF \l_@@_submatrix_name_str
8250 { \@@_node_right:nnnn #2 { } { #3 } { #4 } }
8251 {
8252     \@@_node_right:nnnn #2
8253     { \@@_env: - \l_@@_submatrix_name_str - right } { #3 } { #4 }
8254 }
8255 \cs_set_eq:NN \pgfpointanchor \@@_pgfpointanchor:n
8256 \flag_clear_new:n { nicematrix }
8257 \l_@@_code_tl
8258 }

```

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 $i-j$, $row-i$, $col-j$ 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`.

```

8259 \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.

```

8260 \cs_new_protected:Npn \@@_pgfpointanchor:n #1
8261   {
8262     \use:e
8263     { \exp_not:N \@@_old_pgfpointanchor { \@@_pgfpointanchor_i:nn #1 } }
8264   }

```

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 -.

```

8265 \cs_new:Npn \@@_pgfpointanchor_i:nn #1 #2
8266   { #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.

```

8267 \tl_const:Nn \c_@@_integers_alist_tl
8268   {
8269     { 1 } { } { 2 } { } { 3 } { } { 4 } { } { 5 } { }
8270     { 6 } { } { 7 } { } { 8 } { } { 9 } { } { 10 } { }
8271     { 11 } { } { 12 } { } { 13 } { } { 14 } { } { 15 } { }
8272     { 16 } { } { 17 } { } { 18 } { } { 19 } { } { 20 } { }
8273   }

```

```

8274 \cs_new:Npn \@@_pgfpointanchor_ii:w #1-#2\q_stop
8275   {

```

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`.

```

8276   \tl_if_empty:nTF { #2 }
8277     {
8278       \str_case:nVTF { #1 } \c_@@_integers_alist_tl
8279         {
8280           \flag_raise:n { nicematrix }
8281           \int_if_even:nTF { \flag_height:n { nicematrix } }
8282             { \int_eval:n { #1 + \l_@@_first_i_tl - 1 } }
8283             { \int_eval:n { #1 + \l_@@_first_j_tl - 1 } }
8284         }
8285       { #1 }
8286     }

```

If there is an hyphen, we have to see whether we have a node of the form $i-j$, $row-i$ or $col-j$.

```

8287     { \@@_pgfpointanchor_iii:w { #1 } #2 }
8288   }

```

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`).

```

8289 \cs_new:Npn \@@_pgfpointanchor_iii:w #1 #2 -
8290   {
8291     \str_case:nnF { #1 }
8292       {
8293         { row } { row - \int_eval:n { #2 + \l_@@_first_i_tl - 1 } }
8294         { col } { col - \int_eval:n { #2 + \l_@@_first_j_tl - 1 } }
8295       }

```

Now the case of a node of the form $i-j$.

```

8296     {
8297         \int_eval:n { #1 + \l_@@_first_i_tl - 1 }
8298         - \int_eval:n { #2 + \l_@@_first_j_tl - 1 }
8299     }
8300 }

```

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`).

```

8301 \cs_new_protected:Npn \@@_node_left:nn #1 #2
8302 {
8303     \pgfnode
8304     { rectangle }
8305     { east }
8306     {
8307         \nullfont
8308         \c_math_toggle_token
8309         \@@_color:V \l_@@_delimiters_color_tl
8310         \left #1
8311         \vcenter
8312         {
8313             \nullfont
8314             \hrule \@height \l_tmpa_dim
8315                 \@depth \c_zero_dim
8316                 \@width \c_zero_dim
8317         }
8318         \right .
8319         \c_math_toggle_token
8320     }
8321     { #2 }
8322     { }
8323 }

```

The command `\@@_node_right:nnn` puts the right delimiter with the correct size. The argument #1 is the delimiter to put. The argument #2 is the name we will give to this PGF node (if the key `name` has been used in `\SubMatrix`). The argument #3 is the subscript and #4 is the superscript.

```

8324 \cs_new_protected:Npn \@@_node_right:nnn #1 #2 #3 #4
8325 {
8326     \pgfnode
8327     { rectangle }
8328     { west }
8329     {
8330         \nullfont
8331         \c_math_toggle_token
8332         \@@_color:V \l_@@_delimiters_color_tl
8333         \left .
8334         \vcenter
8335         {
8336             \nullfont
8337             \hrule \@height \l_tmpa_dim
8338                 \@depth \c_zero_dim
8339                 \@width \c_zero_dim
8340         }
8341         \right #1
8342         \tl_if_empty:nF { #3 } { _ { \smash { #3 } } }
8343         ^ { \smash { #4 } }
8344         \c_math_toggle_token
8345     }
8346     { #2 }
8347     { }
8348 }

```

35 Les commandes `\UnderBrace` et `\OverBrace`

The following commands will be linked to `\UnderBrace` and `\OverBrace` in the `\CodeAfter`.

```

8349 \NewDocumentCommand \@@_UnderBrace { 0 { } m m m 0 { } }
8350 {
8351   \peek_remove_spaces:n
8352   { \@@_brace:nnnn { #2 } { #3 } { #4 } { #1 , #5 } { under } }
8353 }

8354 \NewDocumentCommand \@@_OverBrace { 0 { } m m m 0 { } }
8355 {
8356   \peek_remove_spaces:n
8357   { \@@_brace:nnnn { #2 } { #3 } { #4 } { #1 , #5 } { over } }
8358 }

8359 \keys_define:nn { NiceMatrix / Brace }
8360 {
8361   left-shorten .bool_set:N = \l_@@_brace_left_shorten_bool ,
8362   left-shorten .default:n = true ,
8363   right-shorten .bool_set:N = \l_@@_brace_right_shorten_bool ,
8364   shorten .meta:n = { left-shorten , right-shorten } ,
8365   right-shorten .default:n = true ,
8366   yshift .dim_set:N = \l_@@_brace_yshift_dim ,
8367   yshift .value_required:n = true ,
8368   yshift .initial:n = \c_zero_dim ,
8369   color .tl_set:N = \l_tmpa_tl ,
8370   color .value_required:n = true ,
8371   unknown .code:n = \@@_error:n { Unknown-key-for-Brace }
8372 }

```

`#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`.

```

8373 \cs_new_protected:Npn \@@_brace:nnnn #1 #2 #3 #4 #5
8374 {
8375   \group_begin:

```

The four following token lists correspond to the position of the sub-matrix to which a brace will be attached.

```

8376 \@@_compute_i_j:nn { #1 } { #2 }
8377 \bool_lazy_or:nnTF
8378 { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
8379 { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
8380 {
8381   \str_if_eq:nnTF { #5 } { under }
8382   { \@@_error:nn { Construct-too-large } { \UnderBrace } }
8383   { \@@_error:nn { Construct-too-large } { \OverBrace } }
8384 }
8385 {
8386   \tl_clear:N \l_tmpa_tl
8387   \keys_set:nn { NiceMatrix / Brace } { #4 }
8388   \tl_if_empty:NF \l_tmpa_tl { \color { \l_tmpa_tl } }
8389   \pgfpicture
8390   \pgfrememberpicturerepositiononpagetrue
8391   \pgf@relevantforpicturesizefalse
8392   \bool_if:NT \l_@@_brace_left_shorten_bool
8393   {
8394     \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
8395     \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
8396     {
8397       \cs_if_exist:cT
8398       { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
8399       {

```

```

8400         \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
8401         \dim_set:Nn \l_@@_x_initial_dim
8402             { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
8403     }
8404 }
8405 }
8406 \bool_lazy_or:nnT
8407 { \bool_not_p:n \l_@@_brace_left_shorten_bool }
8408 { \dim_compare_p:nNn \l_@@_x_initial_dim = \c_max_dim }
8409 {
8410     \@@_qpoint:n { col - \l_@@_first_j_tl }
8411     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
8412 }
8413 \bool_if:NT \l_@@_brace_right_shorten_bool
8414 {
8415     \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
8416     \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
8417     {
8418         \cs_if_exist:cT
8419             { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
8420         {
8421             \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
8422             \dim_set:Nn \l_@@_x_final_dim
8423                 { \dim_max:nn \l_@@_x_final_dim \pgf@x }
8424         }
8425     }
8426 }
8427 \bool_lazy_or:nnT
8428 { \bool_not_p:n \l_@@_brace_right_shorten_bool }
8429 { \dim_compare_p:nNn \l_@@_x_final_dim = { - \c_max_dim } }
8430 {
8431     \@@_qpoint:n { col - \int_eval:n { \l_@@_last_j_tl + 1 } }
8432     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
8433 }
8434 \pgfset { inner-sep = \c_zero_dim }
8435 \str_if_eq:nnTF { #5 } { under }
8436 { \@@_underbrace_i:n { #3 } }
8437 { \@@_overbrace_i:n { #3 } }
8438 \endpgfpicture
8439 }
8440 \group_end:
8441 }

```

The argument is the text to put above the brace.

```

8442 \cs_new_protected:Npn \@@_overbrace_i:n #1
8443 {
8444     \@@_qpoint:n { row - \l_@@_first_i_tl }
8445     \pgftransformshift
8446     {
8447         \pgfpoint
8448             { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
8449             { \pgf@y + \l_@@_brace_yshift_dim - 3 pt }
8450     }
8451     \pgfnode
8452     { rectangle }
8453     { south }
8454     {
8455         \vbox_top:n
8456         {
8457             \group_begin:
8458             \everycr { }
8459             \halign
8460             {
8461                 \hfil ## \hfil \crcr

```



```

8462         \@@_math_toggle_token: #1 \@@_math_toggle_token: \cr
8463         \noalign { \skip_vertical:n { 3 pt } \nointerlineskip }
8464         \c_math_toggle_token
8465         \overbrace
8466         {
8467             \hbox_to_wd:nn
8468             { \l_@@_x_final_dim - \l_@@_x_initial_dim }
8469             { }
8470         }
8471         \c_math_toggle_token
8472         \cr
8473     }
8474 \group_end:
8475 }
8476 }
8477 { }
8478 { }
8479 }

```

The argument is the text to put under the brace.

```

8480 \cs_new_protected:Npn \@@_underbrace_i:n #1
8481 {
8482     \@@_qpoint:n { row - \int_eval:n { \l_@@_last_i_tl + 1 } }
8483     \pgftransformshift
8484     {
8485         \pgfpoint
8486         { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
8487         { \pgf@y - \l_@@_brace_yshift_dim + 3 pt }
8488     }
8489     \pgfnode
8490     { rectangle }
8491     { north }
8492     {
8493         \group_begin:
8494         \everycr { }
8495         \vbox:n
8496         {
8497             \halign
8498             {
8499                 \hfil ## \hfil \crcr
8500                 \c_math_toggle_token
8501                 \underbrace
8502                 {
8503                     \hbox_to_wd:nn
8504                     { \l_@@_x_final_dim - \l_@@_x_initial_dim }
8505                     { }
8506                 }
8507                 \c_math_toggle_token
8508                 \cr
8509                 \noalign { \skip_vertical:n { 3 pt } \nointerlineskip }
8510                 \@@_math_toggle_token: #1 \@@_math_toggle_token: \cr
8511             }
8512         }
8513     \group_end:
8514 }
8515 { }
8516 { }
8517 }

```

36 The command \ShowCellNames

```

8518 \NewDocumentCommand \@@_ShowCellNames_CodeBefore { }
8519 {
8520   \dim_zero_new:N \g_@@_tmpc_dim
8521   \dim_zero_new:N \g_@@_tmpd_dim
8522   \dim_zero_new:N \g_@@_tmpe_dim
8523   \int_step_inline:nn \c@iRow
8524   {
8525     \begin { pgfpicture }
8526     \@@_qpoint:n { row - ##1 }
8527     \dim_set_eq:NN \l_tmpa_dim \pgf@y
8528     \@@_qpoint:n { row - \int_eval:n { ##1 + 1 } }
8529     \dim_gset:Nn \g_tmpa_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
8530     \dim_gset:Nn \g_tmpb_dim { \l_tmpa_dim - \pgf@y }
8531     \bool_if:NTF \l_@@_in_code_after_bool
8532     \end { pgfpicture }
8533     \int_step_inline:nn \c@jCol
8534     {
8535       \hbox_set:Nn \l_tmpa_box
8536       { \normalfont \Large \color { red ! 50 } ##1 - #####1 }
8537       \begin { pgfpicture }
8538       \@@_qpoint:n { col - #####1 }
8539       \dim_gset_eq:NN \g_@@_tmpc_dim \pgf@x
8540       \@@_qpoint:n { col - \int_eval:n { #####1 + 1 } }
8541       \dim_gset:Nn \g_@@_tmpd_dim { \pgf@x - \g_@@_tmpc_dim }
8542       \dim_gset_eq:NN \g_@@_tmpe_dim \pgf@x
8543       \endpgfpicture
8544       \end { pgfpicture }
8545       \fp_set:Nn \l_tmpa_fp
8546       {
8547         \fp_min:nn
8548         {
8549           \fp_min:nn
8550           { \dim_ratio:nn { \g_@@_tmpd_dim } { \box_wd:N \l_tmpa_box } }
8551           { \dim_ratio:nn { \g_tmpb_dim } { \box_ht_plus_dp:N \l_tmpa_box } }
8552         }
8553         { 1.0 }
8554       }
8555       \box_scale:Nnn \l_tmpa_box { \fp_use:N \l_tmpa_fp } { \fp_use:N \l_tmpa_fp }
8556       \pgfpicture
8557       \pgfrememberpicturepositiononpagetrue
8558       \pgf@relevantforpicturesizefalse
8559       \pgftransformshift
8560       {
8561         \pgfpoint
8562         { 0.5 * ( \g_@@_tmpc_dim + \g_@@_tmpe_dim ) }
8563         { \dim_use:N \g_tmpa_dim }
8564       }
8565       \pgfnode
8566       { rectangle }
8567       { center }
8568       { \box_use:N \l_tmpa_box }
8569       { }
8570       { }
8571       \endpgfpicture
8572     }
8573   }
8574 }
8575 \NewDocumentCommand \@@_ShowCellNames { }
8576 {
8577   \bool_if:NT \l_@@_in_code_after_bool

```

```

8578 {
8579   \pgfpicture
8580   \pgfrememberpicturepositiononpagetrue
8581   \pgf@relevantforpicturesizefalse
8582   \pgfpathrectanglecorners
8583     { \@@_qpoint:n { 1 } }
8584     {
8585       \@@_qpoint:n
8586       { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
8587     }
8588   \pgfsetfillopacity { 0.75 }
8589   \pgfsetfillcolor { white }
8590   \pgfusepathqfill
8591   \endpgfpicture
8592 }
8593 \dim_zero_new:N \g_@@_tmpc_dim
8594 \dim_zero_new:N \g_@@_tmpd_dim
8595 \dim_zero_new:N \g_@@_tmpe_dim
8596 \int_step_inline:nn \c@iRow
8597 {
8598   \bool_if:NTF \l_@@_in_code_after_bool
8599   {
8600     \pgfpicture
8601     \pgfrememberpicturepositiononpagetrue
8602     \pgf@relevantforpicturesizefalse
8603   }
8604   { \begin { pgfpicture } }
8605   \@@_qpoint:n { row - ##1 }
8606   \dim_set_eq:NN \l_tmpa_dim \pgf@y
8607   \@@_qpoint:n { row - \int_eval:n { ##1 + 1 } }
8608   \dim_gset:Nn \g_tmpa_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
8609   \dim_gset:Nn \g_tmpb_dim { \l_tmpa_dim - \pgf@y }
8610   \bool_if:NTF \l_@@_in_code_after_bool
8611     { \endpgfpicture }
8612     { \end { pgfpicture } }
8613   \int_step_inline:nn \c@jCol
8614   {
8615     \hbox_set:Nn \l_tmpa_box
8616     {
8617       \normalfont \Large \sffamily \bfseries
8618       \bool_if:NTF \l_@@_in_code_after_bool
8619         { \color { red } }
8620         { \color { red ! 50 } }
8621       ##1 - ####1
8622     }
8623     \bool_if:NTF \l_@@_in_code_after_bool
8624     {
8625       \pgfpicture
8626       \pgfrememberpicturepositiononpagetrue
8627       \pgf@relevantforpicturesizefalse
8628     }
8629     { \begin { pgfpicture } }
8630     \@@_qpoint:n { col - ####1 }
8631     \dim_gset_eq:NN \g_@@_tmpc_dim \pgf@x
8632     \@@_qpoint:n { col - \int_eval:n { ####1 + 1 } }
8633     \dim_gset:Nn \g_@@_tmpd_dim { \pgf@x - \g_@@_tmpc_dim }
8634     \dim_gset_eq:NN \g_@@_tmpe_dim \pgf@x
8635     \bool_if:NTF \l_@@_in_code_after_bool
8636       { \endpgfpicture }
8637       { \end { pgfpicture } }
8638     \fp_set:Nn \l_tmpa_fp
8639     {
8640       \fp_min:nn

```

```

8641     {
8642         \fp_min:nn
8643         { \dim_ratio:nn { \g_@@_tmpd_dim } { \box_wd:N \l_tmpa_box } }
8644         { \dim_ratio:nn { \g_tmpb_dim } { \box_ht_plus_dp:N \l_tmpa_box } }
8645     }
8646     { 1.0 }
8647 }
8648 \box_scale:Nnn \l_tmpa_box { \fp_use:N \l_tmpa_fp } { \fp_use:N \l_tmpa_fp }
8649 \pgfpicture
8650 \pgfrememberpicturepositiononpagetrue
8651 \pgf@relevantforpicturesizefalse
8652 \pgftransformshift
8653 {
8654     \pgfpoint
8655     { 0.5 * ( \g_@@_tmpc_dim + \g_@@_tmpe_dim ) }
8656     { \dim_use:N \g_tmpa_dim }
8657 }
8658 \pgfnode
8659 { rectangle }
8660 { center }
8661 { \box_use:N \l_tmpa_box }
8662 { }
8663 { }
8664 \endpgfpicture
8665 }
8666 }
8667 }

```

37 We process the options at package loading

We process the options when the package is loaded (with `\usepackage`) but we recommend to use `\NiceMatrixOptions` instead.

We must process these options after the definition of the environment `{NiceMatrix}` because the option `renew-matrix` executes the code `\cs_set_eq:NN \env@matrix \NiceMatrix`.

Of course, the command `\NiceMatrix` must be defined before such an instruction is executed.

The boolean `\g_@@_footnotehyper_bool` will indicate if the option `footnotehyper` is used.

```
8668 \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.

```

8669 \bool_new:N \g_@@_footnote_bool
8670 \msg_new:nnnn { nicematrix } { Unknown-key-for-package }
8671 {
8672     The~key~'\l_keys_key_str'-is-unknown. \\
8673     That~key~will~be~ignored. \\
8674     For~a~list~of~the~available~keys,~type~H~<return>.
8675 }
8676 {
8677     The~available~keys~are~(in~alphabetic~order):~
8678     footnote,~
8679     footnotehyper,~
8680     messages-for-Overleaf,~
8681     no-test-for-array,~
8682     renew-dots,~and
8683     renew-matrix.
8684 }
8685 \keys_define:nn { NiceMatrix / Package }
8686 {
8687     renew-dots .bool_set:N = \l_@@_renew_dots_bool ,

```

```

8688   renew-dots .value_forbidden:n = true ,
8689   renew-matrix .code:n = \@@_renew_matrix: ,
8690   renew-matrix .value_forbidden:n = true ,
8691   messages-for-Overleaf .bool_set:N = \g_@@_messages_for_Overleaf_bool ,
8692   footnote .bool_set:N = \g_@@_footnote_bool ,
8693   footnotehyper .bool_set:N = \g_@@_footnotehyper_bool ,
8694   no-test-for-array .bool_set:N = \g_@@_no_test_for_array_bool ,
8695   no-test-for-array .default:n = true ,
8696   unknown .code:n = \@@_error:n { Unknown-key-for-package }
8697 }
8698 \ProcessKeysOptions { NiceMatrix / Package }

8699 \@@_msg_new:nn { footnote~with~footnotehyper~package }
8700 {
8701   You~can't~use~the~option~'footnote'~because~the~package~
8702   footnotehyper~has~already~been~loaded.~
8703   If~you~want,~you~can~use~the~option~'footnotehyper'~and~the~footnotes~
8704   within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
8705   of~the~package~footnotehyper.\\
8706   The~package~footnote~won't~be~loaded.
8707 }

8708 \@@_msg_new:nn { footnotehyper~with~footnote~package }
8709 {
8710   You~can't~use~the~option~'footnotehyper'~because~the~package~
8711   footnote~has~already~been~loaded.~
8712   If~you~want,~you~can~use~the~option~'footnote'~and~the~footnotes~
8713   within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
8714   of~the~package~footnote.\\
8715   The~package~footnotehyper~won't~be~loaded.
8716 }

```

```

8717 \bool_if:NT \g_@@_footnote_bool
8718 {

```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```

8719   \IfClassLoadedTF { beamer }
8720     { \bool_set_false:N \g_@@_footnote_bool }
8721     {
8722       \IfPackageLoadedTF { footnotehyper }
8723         { \@@_error:n { footnote~with~footnotehyper~package } }
8724         { \usepackage { footnote } }
8725     }
8726 }

```

```

8727 \bool_if:NT \g_@@_footnotehyper_bool
8728 {

```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```

8729   \IfClassLoadedTF { beamer }
8730     { \bool_set_false:N \g_@@_footnote_bool }
8731     {
8732       \IfPackageLoadedTF { footnote }
8733         { \@@_error:n { footnotehyper~with~footnote~package } }
8734         { \usepackage { footnotehyper } }
8735     }
8736   \bool_set_true:N \g_@@_footnote_bool
8737 }

```

The flag `\g_@@_footnote_bool` is raised and so, we will only have to test `\g_@@_footnote_bool` in order to know if we have to insert an environment `{savenotes}`.

38 About the package underscore

If the user loads the package underscore, it must be loaded *before* the package nicematrix. If it is loaded after, we raise an error.

```
8738 \bool_new:N \l_@@_underscore_loaded_bool
8739 \IfPackageLoadedTF { underscore }
8740 { \bool_set_true:N \l_@@_underscore_loaded_bool }
8741 { }

8742 \hook_gput_code:nnn { begindocument } { . }
8743 {
8744   \bool_if:NF \l_@@_underscore_loaded_bool
8745   {
8746     \IfPackageLoadedTF { underscore }
8747     { \@@_error:n { underscore~after~nicematrix } }
8748     { }
8749   }
8750 }
```

39 Error messages of the package

```
8751 \bool_if:NTF \g_@@_messages_for_Overleaf_bool
8752 { \str_const:Nn \c_@@_available_keys_str { } }
8753 {
8754   \str_const:Nn \c_@@_available_keys_str
8755   { For~a~list~of~the~available~keys,~type-H~<return>. }
8756 }

8757 \seq_new:N \g_@@_types_of_matrix_seq
8758 \seq_gset_from_clist:Nn \g_@@_types_of_matrix_seq
8759 {
8760   NiceMatrix ,
8761   pNiceMatrix , bNiceMatrix , vNiceMatrix, BNiceMatrix, VNiceMatrix
8762 }
8763 \seq_gset_map_x:NNn \g_@@_types_of_matrix_seq \g_@@_types_of_matrix_seq
8764 { \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:NVTF` 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`.

```
8765 \cs_new_protected:Npn \@@_error_too_much_cols:
8766 {
8767   \seq_if_in:NVTF \g_@@_types_of_matrix_seq \g_@@_name_env_str
8768   {
8769     \int_compare:nNnTF \l_@@_last_col_int = { -2 }
8770     { \@@_fatal:n { too-much-cols-for-matrix } }
8771     {
8772       \int_compare:nNnTF \l_@@_last_col_int = { -1 }
8773       { \@@_fatal:n { too-much-cols-for-matrix } }
8774       {
8775         \bool_if:NF \l_@@_last_col_without_value_bool
8776         { \@@_fatal:n { too-much-cols-for-matrix-with-last-col } }
8777       }
8778     }
8779   }
8780   {
8781     \IfPackageLoadedTF { tabularx }
```

```

8782     {
8783         \str_if_eq:VnTF \g_@@_name_env_str { NiceTabularX }
8784         {
8785             \int_compare:nNnTF \c@iRow = \c_zero_int
8786             { \@@_fatal:n { X~columns~with~tabularx } }
8787             {
8788                 \@@_fatal:nn { too-much-cols-for-array }
8789                 {
8790                     However,~this~message~may~be~erroneous:~
8791                     maybe~you~have~used~X~columns~while~'tabularx'~is~loaded,~
8792                     ~which~is~forbidden~(however,~it's~still~possible~to~use~
8793                     X~columns~in~{NiceTabularX}).
8794                 }
8795             }
8796         }
8797         { \@@_fatal:nn { too-much-cols-for-array } { } }
8798     }
8799     { \@@_fatal:nn { too-much-cols-for-array } { } }
8800 }
8801 }

```

The following command must *not* be protected since it's used in an error message.

```

8802 \cs_new:Npn \@@_message_hdotsfor:
8803 {
8804     \tl_if_empty:VF \g_@@_HVdotsfor_lines_tl
8805     { ~Maybe~your~use~of~\token_to_str:N \Hdotsfor\ is~incorrect.}
8806 }
8807 \@@_msg_new:nn { hvlines,~rounded-corners~and~corners }
8808 {
8809     Incompatible~options.\\
8810     You~should~not~use~'hvlines',~'rounded-corners'~and~'corners'~at~this~time.\\
8811     The~output~will~not~be~reliable.
8812 }
8813 \@@_msg_new:nn { negative~weight }
8814 {
8815     Negative~weight.\\
8816     The~weight~of~the~'X'~columns~must~be~positive~and~you~have~used~
8817     the~value~'\int_use:N \l_@@_weight_int'.\\
8818     The~absolute~value~will~be~used.
8819 }
8820 \@@_msg_new:nn { last~col~not~used }
8821 {
8822     Column~not~used.\\
8823     The~key~'last-col'~is~in~force~but~you~have~not~used~that~last~column~
8824     in~your~\@@_full_name_env:~.~However,~you~can~go~on.
8825 }
8826 \@@_msg_new:nn { too-much~cols~for~matrix~with~last~col }
8827 {
8828     Too~much~columns.\\
8829     In~the~row~\int_eval:n { \c@iRow },~
8830     you~try~to~use~more~columns~
8831     than~allowed~by~your~\@@_full_name_env:.\@@_message_hdotsfor:\
8832     The~maximal~number~of~columns~is~\int_eval:n { \l_@@_last_col_int - 1 }~
8833     (plus~the~exterior~columns).~This~error~is~fatal.
8834 }
8835 \@@_msg_new:nn { too-much~cols~for~matrix }
8836 {
8837     Too~much~columns.\\
8838     In~the~row~\int_eval:n { \c@iRow },~
8839     you~try~to~use~more~columns~than~allowed~by~your~
8840     \@@_full_name_env:.\@@_message_hdotsfor:\ Recall~that~the~maximal~
8841     number~of~columns~for~a~matrix~(excepted~the~potential~exterior~

```

```

8842 columns)~is~fixed~by~the~LaTeX~counter~'MaxMatrixCols'~.~
8843 Its~current~value~is~\int_use:N \c@MaxMatrixCols\ (use~
8844 \token_to_str:N \setcounter\ to~change~that~value).~
8845 This~error~is~fatal.
8846 }

8847 \@@_msg_new:nn { too-much-cols-for-array }
8848 {
8849   Too~much~columns.\\
8850   In~the~row~\int_eval:n { \c@iRow },~
8851   ~you~try~to~use~more~columns~than~allowed~by~your~
8852   \@@_full_name_env:.\@@_message_hdotsfor:\ The~maximal~number~of~columns~is~
8853   \int_use:N \g_@@_static_num_of_col_int\
8854   ~(plus~the~potential~exterior~ones).~#1
8855   This~error~is~fatal.
8856 }

8857 \@@_msg_new:nn { X-columns-with-tabularx }
8858 {
8859   There~is~a~problem.\\
8860   You~have~probably~used~X~columns~in~your~environment~{\g_@@_name_env_str}.~
8861   That's~not~allowed~because~'tabularx'~is~loaded~(however,~you~can~use~X~columns~
8862   in~an~environment~{NiceTabularX}).\\
8863   This~error~is~fatal.
8864 }

8865 \@@_msg_new:nn { columns-not-used }
8866 {
8867   Columns~not~used.\\
8868   The~preamble~of~your~\@@_full_name_env:\ announces~\int_use:N
8869   \g_@@_static_num_of_col_int\ columns~but~you~use~only~\int_use:N \c@jCol.\\
8870   The~columns~you~did~not~used~won't~be~created.\\
8871   You~won't~have~similar~error~till~the~end~of~the~document.
8872 }

8873 \@@_msg_new:nn { in-first-col }
8874 {
8875   Erroneous~use.\\
8876   You~can't~use~the~command~#1 in~the~first~column~(number~0)~of~the~array.\\
8877   That~command~will~be~ignored.
8878 }

8879 \@@_msg_new:nn { in-last-col }
8880 {
8881   Erroneous~use.\\
8882   You~can't~use~the~command~#1 in~the~last~column~(exterior)~of~the~array.\\
8883   That~command~will~be~ignored.
8884 }

8885 \@@_msg_new:nn { in-first-row }
8886 {
8887   Erroneous~use.\\
8888   You~can't~use~the~command~#1 in~the~first~row~(number~0)~of~the~array.\\
8889   That~command~will~be~ignored.
8890 }

8891 \@@_msg_new:nn { in-last-row }
8892 {
8893   You~can't~use~the~command~#1 in~the~last~row~(exterior)~of~the~array.\\
8894   That~command~will~be~ignored.
8895 }

8896 \@@_msg_new:nn { caption-outside-float }
8897 {
8898   Key~caption~forbidden.\\
8899   You~can't~use~the~key~'caption'~because~you~are~not~in~a~floating~
8900   environment.~This~key~will~be~ignored.

```



```

8901 }
8902 \@@_msg_new:nn { short-caption~without~caption }
8903 {
8904   You~should~not~use~the~key~'short-caption'~without~'caption'.~
8905   However,~your~'short-caption'~will~be~used~as~'caption'.
8906 }
8907 \@@_msg_new:nn { double-closing~delimiter }
8908 {
8909   Double~delimiter.\\
8910   You~can't~put~a~second~closing~delimiter~"#1"~just~after~a~first~closing~
8911   delimiter.~This~delimiter~will~be~ignored.
8912 }
8913 \@@_msg_new:nn { delimiter~after~opening }
8914 {
8915   Double~delimiter.\\
8916   You~can't~put~a~second~delimiter~"#1"~just~after~a~first~opening~
8917   delimiter.~That~delimiter~will~be~ignored.
8918 }
8919 \@@_msg_new:nn { bad~option~for~line~style }
8920 {
8921   Bad~line~style.\\
8922   Since~you~haven't~loaded~Tikz,~the~only~value~you~can~give~to~'line-style'~
8923   is~'standard'.~That~key~will~be~ignored.
8924 }
8925 \@@_msg_new:nn { Identical~notes~in~caption }
8926 {
8927   Identical~tabular~notes.\\
8928   You~can't~put~several~notes~with~the~same~content~in~
8929   \token_to_str:N \caption\ (but~you~can~in~the~main~tabular).\\
8930   If~you~go~on,~the~output~will~probably~be~erroneous.
8931 }
8932 \@@_msg_new:nn { tabularnote~below~the~tabular }
8933 {
8934   \token_to_str:N \tabularnote\ forbidden\\
8935   You~can't~use~\token_to_str:N \tabularnote\ in~the~caption~
8936   of~your~tabular~because~the~caption~will~be~composed~below~
8937   the~tabular.~If~you~want~the~caption~above~the~tabular~use~the~
8938   key~'caption-above'~in~\token_to_str:N \NiceMatrixOptions.\\
8939   Your~\token_to_str:N \tabularnote\ will~be~discarded~and~
8940   no~similar~error~will~raised~in~this~document.
8941 }
8942 \@@_msg_new:nn { Unknown~key~for~rules }
8943 {
8944   Unknown~key.\\
8945   There~is~only~two~keys~available~here:~width~and~color.\\
8946   Your~key~'\l_keys_key_str'~will~be~ignored.
8947 }
8948 \@@_msg_new:nn { Unknown~key~for~rotate }
8949 {
8950   Unknown~key.\\
8951   The~only~key~available~here~is~'c'.\\
8952   Your~key~'\l_keys_key_str'~will~be~ignored.
8953 }
8954 \@@_msg_new:nnn { Unknown~key~for~custom~line }
8955 {
8956   Unknown~key.\\
8957   The~key~'\l_keys_key_str'~is~unknown~in~a~'custom~line'.~
8958   It~you~go~on,~you~will~probably~have~other~errors. \\
8959   \c_@@_available_keys_str
8960 }

```

```

8961 {
8962   The~available~keys~are~(in~alphabetic~order):~
8963   ccommand,~
8964   color,~
8965   command,~
8966   dotted,~
8967   letter,~
8968   multiplicity,~
8969   sep-color,~
8970   tikz,~and~total-width.
8971 }
8972 \@@_msg_new:nnn { Unknown~key~for~xdots }
8973 {
8974   Unknown~key.\\
8975   The~key~'\l_keys_key_str'~is~unknown~for~a~command~for~drawing~dotted~rules.\\
8976   \c_@@_available_keys_str
8977 }
8978 {
8979   The~available~keys~are~(in~alphabetic~order):~
8980   'color',~
8981   'horizontal-labels',~
8982   'inter',~
8983   'line-style',~
8984   'radius',~
8985   'shorten',~
8986   'shorten-end'~and~'shorten-start'.
8987 }
8988 \@@_msg_new:nn { Unknown~key~for~rowcolors }
8989 {
8990   Unknown~key.\\
8991   As~for~now,~there~is~only~two~keys~available~here:~'cols'~and~'respect-blocks'~
8992   (and~you~try~to~use~'\l_keys_key_str')\\
8993   That~key~will~be~ignored.
8994 }
8995 \@@_msg_new:nn { label~without~caption }
8996 {
8997   You~can't~use~the~key~'label'~in~your~'{NiceTabular}'~because~
8998   you~have~not~used~the~key~'caption'.~The~key~'label'~will~be~ignored.
8999 }
9000 \@@_msg_new:nn { W~warning }
9001 {
9002   Line~\msg_line_number:~.~The~cell~is~too~wide~for~your~column~'W'~
9003   (row~\int_use:N \c@iRow).
9004 }
9005 \@@_msg_new:nn { Construct~too~large }
9006 {
9007   Construct~too~large.\\
9008   Your~command~\token_to_str:N #1
9009   can't~be~drawn~because~your~matrix~is~too~small.\\
9010   That~command~will~be~ignored.
9011 }
9012 \@@_msg_new:nn { underscore~after~nicematrix }
9013 {
9014   Problem~with~'underscore'.\\
9015   The~package~'underscore'~should~be~loaded~before~'nicematrix'.~
9016   You~can~go~on~but~you~won't~be~able~to~write~something~such~as:\\
9017   '\token_to_str:N \Cdots\token_to_str:N _{n~\token_to_str:N \text{~times}}'.
9018 }
9019 \@@_msg_new:nn { ampersand~in~light~syntax }
9020 {
9021   Ampersand~forbidden.\\

```

```

9022   You~can't~use~an~ampersand~(\token_to_str:N &~)~to~separate~columns~because~
9023   ~the~key~'light-syntax'~is~in~force.~This~error~is~fatal.
9024   }

9025 \@@_msg_new:nn { double-backslash-in~light-syntax }
9026   {
9027     Double-backslash-forbidden.\\
9028     You~can't~use~\token_to_str:N
9029     \\~to~separate~rows~because~the~key~'light-syntax'~
9030     is~in~force.~You~must~use~the~character~'\l_@@_end_of_row_tl'~
9031     (set~by~the~key~'end-of-row')~.~This~error~is~fatal.
9032   }

9033 \@@_msg_new:nn { hlines-with-color }
9034   {
9035     Incompatible~keys.\\
9036     You~can't~use~the~keys~'hlines',~'vlines'~or~'hvlines'~for~a~
9037     '\token_to_str:N \Block'~when~the~key~'color'~or~'draw'~is~used.\\
9038     Maybe~it~will~possible~in~future~version.\\
9039     Your~key~will~be~discarded.
9040   }

9041 \@@_msg_new:nn { bad-value-for~baseline }
9042   {
9043     Bad~value~for~baseline.\\
9044     The~value~given~to~'baseline'~(\int_use:N \l_tmpa_int)~is~not~
9045     valid.~The~value~must~be~between~\int_use:N \l_@@_first_row_int\ and~
9046     \int_use:N \g_@@_row_total_int\ or~equal~to~'t',~'c'~or~'b'~or~of~
9047     the~form~'line-i'.\\
9048     A~value~of~1~will~be~used.
9049   }

9050 \@@_msg_new:nn { ragged2e~not~loaded }
9051   {
9052     You~have~to~load~'ragged2e'~in~order~to~use~the~key~'\l_keys_key_str'~in~
9053     your~column~'\l_@@_vpos_col_str'~(or~'X').~The~key~'\str_lowercase:V
9054     \l_keys_key_str'~will~be~used~instead.
9055   }

9056 \@@_msg_new:nn { Invalid-name }
9057   {
9058     Invalid~name.\\
9059     You~can't~give~the~name~'\l_keys_value_tl'~to~a~\token_to_str:N
9060     \SubMatrix\ of~your~\@@_full_name_env:.\\
9061     A~name~must~be~accepted~by~the~regular~expression~[A-Za-z][A-Za-z0-9]*.\\
9062     This~key~will~be~ignored.
9063   }

9064 \@@_msg_new:nn { Wrong~line~in~SubMatrix }
9065   {
9066     Wrong~line.\\
9067     You~try~to~draw~a~#1~line~of~number~'#2'~in~a~
9068     \token_to_str:N \SubMatrix\ of~your~\@@_full_name_env:\ but~that~
9069     number~is~not~valid.~It~will~be~ignored.
9070   }

9071 \@@_msg_new:nn { Impossible~delimiter }
9072   {
9073     Impossible~delimiter.\\
9074     It's~impossible~to~draw~the~#1~delimiter~of~your~
9075     \token_to_str:N \SubMatrix\ because~all~the~cells~are~empty~
9076     in~that~column.
9077     \bool_if:NT \l_@@_submatrix_slim_bool
9078     { ~Maybe~you~should~try~without~the~key~'slim'. } \\
9079     This~\token_to_str:N \SubMatrix\ will~be~ignored.
9080   }

9081 \@@_msg_new:nn { width~without~X~columns }

```

```

9082 {
9083   You~have~used~the~key~'width'~but~you~have~put~no~'X'~column.~
9084   That~key~will~be~ignored.
9085 }

9086 \@_msg_new:nn { key~multiplicity~with~dotted }
9087 {
9088   Incompatible~keys. \\
9089   You~have~used~the~key~'multiplicity'~with~the~key~'dotted'~
9090   in~a~'custom-line'.~They~are~incompatible. \\
9091   The~key~'multiplicity'~will~be~discarded.
9092 }

9093 \@_msg_new:nn { empty~environment }
9094 {
9095   Empty~environment.\\
9096   Your~\@_full_name_env:\ is~empty.~This~error~is~fatal.
9097 }

9098 \@_msg_new:nn { No~letter~and~no~command }
9099 {
9100   Erroneous~use.\\
9101   Your~use~of~'custom-line'~is~no~op~since~you~don't~have~used~the~
9102   key~'letter'~(for~a~letter~for~vertical~rules)~nor~the~keys~'command'~or~
9103   '~'ccommand'~(to~draw~horizontal~rules).\\
9104   However,~you~can~go~on.
9105 }

9106 \@_msg_new:nn { Forbidden~letter }
9107 {
9108   Forbidden~letter.\\
9109   You~can't~use~the~letter~'\l_@_letter_str'~for~a~customized~line.\\
9110   It~will~be~ignored.
9111 }

9112 \@_msg_new:nn { Several~letters }
9113 {
9114   Wrong~name.\\
9115   You~must~use~only~one~letter~as~value~for~the~key~'letter'~(and~you~
9116   have~used~'\l_@_letter_str').\\
9117   It~will~be~ignored.
9118 }

9119 \@_msg_new:nn { Delimiter~with~small }
9120 {
9121   Delimiter~forbidden.\\
9122   You~can't~put~a~delimiter~in~the~preamble~of~your~\@_full_name_env:\
9123   because~the~key~'small'~is~in~force.\\
9124   This~error~is~fatal.
9125 }

9126 \@_msg_new:nn { unknown~cell~for~line~in~CodeAfter }
9127 {
9128   Unknown~cell.\\
9129   Your~command~\token_to_str:N\line\{#1\}\{#2\}~in~
9130   the~\token_to_str:N \CodeAfter\ of~your~\@_full_name_env:\
9131   can't~be~executed~because~a~cell~doesn't~exist.\\
9132   This~command~\token_to_str:N \line\ will~be~ignored.
9133 }

9134 \@_msg_new:nnn { Duplicate~name~for~SubMatrix }
9135 {
9136   Duplicate~name.\\
9137   The~name~'#1'~is~already~used~for~a~\token_to_str:N \SubMatrix\
9138   in~this~\@_full_name_env:.\\
9139   This~key~will~be~ignored.\\
9140   \bool_if:NF \g_@_messages_for_Overleaf_bool
9141   { For~a~list~of~the~names~already~used,~type~H~<return>. }

```

```

9142 }
9143 {
9144   The~names~already~defined~in~this~\@@_full_name_env:\ are:~
9145   \seq_use:Nmn \g_@@_submatrix_names_seq { ~and~ } { ,~ } { ~and~ }.
9146 }
9147 \@@_msg_new:nn { r-or-l-with-preamble }
9148 {
9149   Erroneous~use.\\
9150   You~can't~use~the~key~'\l_keys_key_str'~in~your~\@@_full_name_env:..~
9151   You~must~specify~the~alignment~of~your~columns~with~the~preamble~of~
9152   your~\@@_full_name_env:..\\
9153   This~key~will~be~ignored.
9154 }
9155 \@@_msg_new:nn { Hdotsfor~in~col-0 }
9156 {
9157   Erroneous~use.\\
9158   You~can't~use~\token_to_str:N \Hdotsfor\ in~an~exterior~column~of~
9159   the~array.~This~error~is~fatal.
9160 }
9161 \@@_msg_new:nn { bad-corner }
9162 {
9163   Bad~corner.\\
9164   #1~is~an~incorrect~specification~for~a~corner~(in~the~key~
9165   'corners').~The~available~values~are:~NW,~SW,~NE~and~SE.\\
9166   This~specification~of~corner~will~be~ignored.
9167 }
9168 \@@_msg_new:nn { bad-border }
9169 {
9170   Bad~border.\\
9171   \l_keys_key_str\space~is~an~incorrect~specification~for~a~border~
9172   (in~the~key~'borders'~of~the~command~\token_to_str:N \Block).~
9173   The~available~values~are:~left,~right,~top~and~bottom~(and~you~can~
9174   also~use~the~key~'tikz'
9175   \IfPackageLoadedTF { tikz }
9176   { }
9177   {~if~you~load~the~LaTeX~package~'tikz'}).\\
9178   This~specification~of~border~will~be~ignored.
9179 }
9180 \@@_msg_new:nn { tikz-key~without~tikz }
9181 {
9182   Tikz~not~loaded.\\
9183   You~can't~use~the~key~'tikz'~for~the~command~'\token_to_str:N
9184   \Block'~because~you~have~not~loaded~tikz.~
9185   This~key~will~be~ignored.
9186 }
9187 \@@_msg_new:nn { last-col~non-empty~for~NiceArray }
9188 {
9189   Erroneous~use.\\
9190   In~the~\@@_full_name_env:,~you~must~use~the~key~
9191   'last-col'~without~value.\\
9192   However,~you~can~go~on~for~this~time~
9193   (the~value~'\l_keys_value_tl'~will~be~ignored).
9194 }
9195 \@@_msg_new:nn { last-col~non-empty~for~NiceMatrixOptions }
9196 {
9197   Erroneous~use.\\
9198   In~\token_to_str:N \NiceMatrixOptions,~you~must~use~the~key~
9199   'last-col'~without~value.\\
9200   However,~you~can~go~on~for~this~time~
9201   (the~value~'\l_keys_value_tl'~will~be~ignored).
9202 }

```

```

9203 \@@_msg_new:nn { Block-too-large-1 }
9204 {
9205   Block-too-large.\
9206   You~try~to~draw~a~block~in~the~cell~#1~#2~of~your~matrix~but~the~matrix~is~
9207   too~small~for~that~block. \
9208 }
9209 \@@_msg_new:nn { Block-too-large-2 }
9210 {
9211   Block-too-large.\
9212   The~preamble~of~your~\@@_full_name_env:\ announces~\int_use:N
9213   \g_@@_static_num_of_col_int\
9214   columns~but~you~use~only~\int_use:N \c@jCol\ and~that's~why~a~block~
9215   specified~in~the~cell~#1~#2~can't~be~drawn.~You~should~add~some~ampersands~
9216   (&)~at~the~end~of~the~first~row~of~your~
9217   \@@_full_name_env:.\
9218   This~block~and~maybe~others~will~be~ignored.
9219 }
9220 \@@_msg_new:nn { unknown-column-type }
9221 {
9222   Bad~column~type.\
9223   The~column~type~'#1'~in~your~\@@_full_name_env:\
9224   is~unknown. \
9225   This~error~is~fatal.
9226 }
9227 \@@_msg_new:nn { unknown-column-type-S }
9228 {
9229   Bad~column~type.\
9230   The~column~type~'S'~in~your~\@@_full_name_env:\ is~unknown. \
9231   If~you~want~to~use~the~column~type~'S'~of~siunitx,~you~should~
9232   load~that~package. \
9233   This~error~is~fatal.
9234 }
9235 \@@_msg_new:nn { tabularnote-forbidden }
9236 {
9237   Forbidden~command.\
9238   You~can't~use~the~command~\token_to_str:N\ tabularnote\
9239   ~here.~This~command~is~available~only~in~
9240   \{NiceTabular\},~\{NiceTabular*\}~and~\{NiceTabularX\}~or~in~
9241   the~argument~of~a~command~\token_to_str:N \caption\ included~
9242   in~an~environment~{table}. \
9243   This~command~will~be~ignored.
9244 }
9245 \@@_msg_new:nn { borders-forbidden }
9246 {
9247   Forbidden~key.\
9248   You~can't~use~the~key~'borders'~of~the~command~\token_to_str:N \Block\
9249   because~the~option~'rounded-corners'~
9250   is~in~force~with~a~non-zero~value.\
9251   This~key~will~be~ignored.
9252 }
9253 \@@_msg_new:nn { bottomrule-without-booktabs }
9254 {
9255   booktabs~not~loaded.\
9256   You~can't~use~the~key~'tabular/bottomrule'~because~you~haven't~
9257   loaded~'booktabs'.\
9258   This~key~will~be~ignored.
9259 }
9260 \@@_msg_new:nn { enumitem-not-loaded }
9261 {
9262   enumitem~not~loaded.\
9263   You~can't~use~the~command~\token_to_str:N\ tabularnote\

```

```

9264 ~because-you-haven't-loaded~'enumitem'.\\
9265 All~the~commands~\token_to_str:N\tabularnote\ will~be-
9266 ignored~in~the~document.
9267 }

9268 \@@_msg_new:nn { tikz~in~custom~line~without~tikz }
9269 {
9270 Tikz~not~loaded.\\
9271 You~have~used~the~key~'tikz'~in~the~definition~of~a~
9272 customized~line~(with~'custom~line')~but~tikz~is~not~loaded.~
9273 You~can~go~on~but~you~will~have~another~error~if~you~actually~
9274 use~that~custom~line.
9275 }

9276 \@@_msg_new:nn { tikz~in~borders~without~tikz }
9277 {
9278 Tikz~not~loaded.\\
9279 You~have~used~the~key~'tikz'~in~a~key~'borders'~(of~a~
9280 command~\token_to_str:N\Block')~but~tikz~is~not~loaded.~
9281 That~key~will~be~ignored.
9282 }

9283 \@@_msg_new:nn { color~in~custom~line~with~tikz }
9284 {
9285 Erroneous~use.\\
9286 In~a~'custom~line',~you~have~used~both~'tikz'~and~'color',~
9287 which~is~forbidden~(you~should~use~'color'~inside~the~key~'tikz').~
9288 The~key~'color'~will~be~discarded.
9289 }

9290 \@@_msg_new:nn { Wrong~last~row }
9291 {
9292 Wrong~number.\\
9293 You~have~used~'last~row=\int_use:N \l_@@_last_row_int'~but~your~
9294 \@@_full_name_env:\ seems~to~have~\int_use:N \c@iRow \ rows.~
9295 If~you~go~on,~the~value~of~\int_use:N \c@iRow \ will~be~used~for~
9296 last~row.~You~can~avoid~this~problem~by~using~'last~row'~
9297 without~value~(more~compilations~might~be~necessary).
9298 }

9299 \@@_msg_new:nn { Yet~in~env }
9300 {
9301 Nested~environments.\\
9302 Environments~of~nicematrix~can't~be~nested.\\
9303 This~error~is~fatal.
9304 }

9305 \@@_msg_new:nn { Outside~math~mode }
9306 {
9307 Outside~math~mode.\\
9308 The~\@@_full_name_env:\ can~be~used~only~in~math~mode~
9309 (and~not~in~\token_to_str:N \vcenter).\\
9310 This~error~is~fatal.
9311 }

9312 \@@_msg_new:nn { One~letter~allowed }
9313 {
9314 Bad~name.\\
9315 The~value~of~key~'\l_keys_key_str'~must~be~of~length~1.\\
9316 It~will~be~ignored.
9317 }

9318 \@@_msg_new:nn { TabularNote~in~CodeAfter }
9319 {
9320 Environment~{TabularNote}~forbidden.\\
9321 You~must~use~{TabularNote}~at~the~end~of~your~{NiceTabular}~
9322 but~*before*~the~\token_to_str:N \CodeAfter.\\
9323 This~environment~{TabularNote}~will~be~ignored.

```

```

9324 }
9325 \@@_msg_new:nn { varwidth~not~loaded }
9326 {
9327   varwidth~not~loaded.\\
9328   You~can't~use~the~column~type~'V'~because~'varwidth'~is~not~
9329   loaded.\\
9330   Your~column~will~behave~like~'p'.
9331 }
9332 \@@_msg_new:nnn { Unknow~key~for~RulesBis }
9333 {
9334   Unknow~key.\\
9335   Your~key~'\l_keys_key_str'~is~unknown~for~a~rule.\\
9336   \c_@@_available_keys_str
9337 }
9338 {
9339   The~available~keys~are~(in~alphabetic~order):~
9340   color,~
9341   dotted,~
9342   multiplicity,~
9343   sep-color,~
9344   tikz,~and~total~width.
9345 }
9346
9347 \@@_msg_new:nnn { Unknown~key~for~Block }
9348 {
9349   Unknown~key.\\
9350   The~key~'\l_keys_key_str'~is~unknown~for~the~command~\token_to_str:N
9351   \Block.\\ It~will~be~ignored. \\
9352   \c_@@_available_keys_str
9353 }
9354 {
9355   The~available~keys~are~(in~alphabetic~order):~b,~B,~borders,~c,~draw,~fill,~
9356   hlines,~hvlines,~l,~line-width,~name,~opacity,~rounded-corners,~r,~
9357   respect-arraystretch,~t,~T,~tikz,~transparent~and~vlines.
9358 }
9359 \@@_msg_new:nn { Version~of~siunitx~too~old }
9360 {
9361   siunitx~too~old.\\
9362   You~can't~use~'S'~columns~because~your~version~of~'siunitx'~
9363   is~too~old.~You~need~at~least~v3.0.38~and~your~log~file~says:~"siunitx,~
9364   \use:c { ver @ siunitx.sty }". \\
9365   This~error~is~fatal.
9366 }
9367 \@@_msg_new:nnn { Unknown~key~for~Brace }
9368 {
9369   Unknown~key.\\
9370   The~key~'\l_keys_key_str'~is~unknown~for~the~commands~\token_to_str:N
9371   \UnderBrace\ and~\token_to_str:N \OverBrace.\\
9372   It~will~be~ignored. \\
9373   \c_@@_available_keys_str
9374 }
9375 {
9376   The~available~keys~are~(in~alphabetic~order):~color,~left-shorten,~
9377   right-shorten,~shorten~(which~fixes~both~left-shorten~and~
9378   right-shorten)~and~yshift.
9379 }
9380 \@@_msg_new:nnn { Unknown~key~for~CodeAfter }
9381 {
9382   Unknown~key.\\
9383   The~key~'\l_keys_key_str'~is~unknown.\\
9384   It~will~be~ignored. \\

```



```

9385 \c_@@_available_keys_str
9386 }
9387 {
9388 The~available~keys~are~(in~alphabetic~order):~
9389 delimiters/color,~
9390 rules~(with~the~subkeys~'color'~and~'width'),~
9391 sub-matrix~(several~subkeys)~
9392 and~xdots~(several~subkeys).~
9393 The~latter~is~for~the~command~\token_to_str:N \line.
9394 }
9395 \@@_msg_new:nnn { Unknown~key~for~CodeBefore }
9396 {
9397 Unknown~key.\\
9398 The~key~'\l_keys_key_str'~is~unknown.\\
9399 It~will~be~ignored. \\
9400 \c_@@_available_keys_str
9401 }
9402 {
9403 The~available~keys~are~(in~alphabetic~order):~
9404 create-cell-nodes,~
9405 delimiters/color~and~
9406 sub-matrix~(several~subkeys).
9407 }
9408 \@@_msg_new:nnn { Unknown~key~for~SubMatrix }
9409 {
9410 Unknown~key.\\
9411 The~key~'\l_keys_key_str'~is~unknown.\\
9412 That~key~will~be~ignored. \\
9413 \c_@@_available_keys_str
9414 }
9415 {
9416 The~available~keys~are~(in~alphabetic~order):~
9417 'delimiters/color',~
9418 'extra-height',~
9419 'hlines',~
9420 'hvlines',~
9421 'left-xshift',~
9422 'name',~
9423 'right-xshift',~
9424 'rules'~(with~the~subkeys~'color'~and~'width'),~
9425 'slim',~
9426 'vlines'~and~'xshift'~(which~sets~both~'left-xshift'~
9427 and~'right-xshift').\\
9428 }
9429 \@@_msg_new:nnn { Unknown~key~for~notes }
9430 {
9431 Unknown~key.\\
9432 The~key~'\l_keys_key_str'~is~unknown.\\
9433 That~key~will~be~ignored. \\
9434 \c_@@_available_keys_str
9435 }
9436 {
9437 The~available~keys~are~(in~alphabetic~order):~
9438 bottomrule,~
9439 code-after,~
9440 code-before,~
9441 detect-duplicates,~
9442 enumitem-keys,~
9443 enumitem-keys-para,~
9444 para,~
9445 label-in-list,~
9446 label-in-tabular~and~
9447 style.

```

```

9448 }
9449 \@_msg_new:nnn { Unknown~key~for~RowStyle }
9450 {
9451   Unknown~key.\\
9452   The~key~'\l_keys_key_str'~is~unknown~for~the~command~
9453   \token_to_str:N \RowStyle. \\
9454   That~key~will~be~ignored. \\
9455   \c_@@_available_keys_str
9456 }
9457 {
9458   The~available~keys~are~(in~alphabetic~order):~
9459   'bold',~
9460   'cell-space-top-limit',~
9461   'cell-space-bottom-limit',~
9462   'cell-space-limits',~
9463   'color',~
9464   'nb-rows'~and~
9465   'rowcolor'.
9466 }
9467 \@_msg_new:nnn { Unknown~key~for~NiceMatrixOptions }
9468 {
9469   Unknown~key.\\
9470   The~key~'\l_keys_key_str'~is~unknown~for~the~command~
9471   \token_to_str:N \NiceMatrixOptions. \\
9472   That~key~will~be~ignored. \\
9473   \c_@@_available_keys_str
9474 }
9475 {
9476   The~available~keys~are~(in~alphabetic~order):~
9477   allow-duplicate-names,~
9478   caption-above,~
9479   cell-space-bottom-limit,~
9480   cell-space-limits,~
9481   cell-space-top-limit,~
9482   code-for-first-col,~
9483   code-for-first-row,~
9484   code-for-last-col,~
9485   code-for-last-row,~
9486   corners,~
9487   custom-key,~
9488   create-extra-nodes,~
9489   create-medium-nodes,~
9490   create-large-nodes,~
9491   delimiters~(several~subkeys),~
9492   end-of-row,~
9493   first-col,~
9494   first-row,~
9495   hlines,~
9496   hvlines,~
9497   hvlines-except-borders,~
9498   last-col,~
9499   last-row,~
9500   left-margin,~
9501   light-syntax,~
9502   matrix/columns-type,~
9503   notes~(several~subkeys),~
9504   nullify-dots,~
9505   pgf-node-code,~
9506   renew-dots,~
9507   renew-matrix,~
9508   respect-arraystretch,~
9509   rounded-corners,~
9510   right-margin,~

```

```

9511 rules~(with~the~subkeys~'color'~and~'width'),~
9512 small,~
9513 sub-matrix~(several~subkeys),~
9514 vlines,~
9515 xdots~(several~subkeys).
9516 }

```

For `{NiceArray}`, the set of keys is the same as for `{NiceMatrix}` excepted that there is no `l` and `r`.

```

9517 \@@_msg_new:nnn { Unknown~key~for~NiceArray }
9518 {
9519   Unknown~key.\\
9520   The~key~'\l_keys_key_str'~is~unknown~for~the~environment~
9521   \{NiceArray\}. \\
9522   That~key~will~be~ignored. \\
9523   \c_@@_available_keys_str
9524 }
9525 {
9526   The~available~keys~are~(in~alphabetic~order):~
9527   b,~
9528   baseline,~
9529   c,~
9530   cell-space-bottom-limit,~
9531   cell-space-limits,~
9532   cell-space-top-limit,~
9533   code-after,~
9534   code-for-first-col,~
9535   code-for-first-row,~
9536   code-for-last-col,~
9537   code-for-last-row,~
9538   color-inside,~
9539   columns-width,~
9540   corners,~
9541   create-extra-nodes,~
9542   create-medium-nodes,~
9543   create-large-nodes,~
9544   extra-left-margin,~
9545   extra-right-margin,~
9546   first-col,~
9547   first-row,~
9548   hlines,~
9549   hvlines,~
9550   hvlines-except-borders,~
9551   last-col,~
9552   last-row,~
9553   left-margin,~
9554   light-syntax,~
9555   name,~
9556   nullify-dots,~
9557   pgf-node-code,~
9558   renew-dots,~
9559   respect-arraystretch,~
9560   right-margin,~
9561   rounded-corners,~
9562   rules~(with~the~subkeys~'color'~and~'width'),~
9563   small,~
9564   t,~
9565   vlines,~
9566   xdots/color,~
9567   xdots/shorten-start,~
9568   xdots/shorten-end,~
9569   xdots/shorten~and~
9570   xdots/line-style.
9571 }

```

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).

```

9572 \@_msg_new:nnn { Unknown~key~for~NiceMatrix }
9573   {
9574     Unknown~key.\\
9575     The~key~'\l_keys_key_str'~is~unknown~for~the~
9576     \@_full_name_env:. \\
9577     That~key~will~be~ignored. \\
9578     \c_@_available_keys_str
9579   }
9580   {
9581     The~available~keys~are~(in~alphabetic~order):~
9582     b,~
9583     baseline,~
9584     c,~
9585     cell-space-bottom-limit,~
9586     cell-space-limits,~
9587     cell-space-top-limit,~
9588     code-after,~
9589     code-for-first-col,~
9590     code-for-first-row,~
9591     code-for-last-col,~
9592     code-for-last-row,~
9593     color-inside,~
9594     columns-type,~
9595     columns-width,~
9596     corners,~
9597     create-extra-nodes,~
9598     create-medium-nodes,~
9599     create-large-nodes,~
9600     extra-left-margin,~
9601     extra-right-margin,~
9602     first-col,~
9603     first-row,~
9604     hlines,~
9605     hvlines,~
9606     hvlines-except-borders,~
9607     l,~
9608     last-col,~
9609     last-row,~
9610     left-margin,~
9611     light-syntax,~
9612     name,~
9613     nullify-dots,~
9614     pgf-node-code,~
9615     r,~
9616     renew-dots,~
9617     respect-arraystretch,~
9618     right-margin,~
9619     rounded-corners,~
9620     rules~(with~the~subkeys~'color'~and~'width'),~
9621     small,~
9622     t,~
9623     vlines,~
9624     xdots/color,~
9625     xdots/shorten-start,~
9626     xdots/shorten-end,~
9627     xdots/shorten~and~
9628     xdots/line-style.
9629   }
9630 \@_msg_new:nnn { Unknown~key~for~NiceTabular }
9631   {
9632     Unknown~key.\\

```

```

9633 The~key~'\l_keys_key_str'~is~unknown~for~the~environment~
9634 \{NiceTabular\}. \\
9635 That~key~will~be~ignored. \\
9636 \c_@@_available_keys_str
9637 }
9638 {
9639 The~available~keys~are~(in~alphabetic~order):~
9640 b,~
9641 baseline,~
9642 c,~
9643 caption,~
9644 cell-space-bottom-limit,~
9645 cell-space-limits,~
9646 cell-space-top-limit,~
9647 code-after,~
9648 code-for-first-col,~
9649 code-for-first-row,~
9650 code-for-last-col,~
9651 code-for-last-row,~
9652 color-inside,~
9653 columns-width,~
9654 corners,~
9655 custom-line,~
9656 create-extra-nodes,~
9657 create-medium-nodes,~
9658 create-large-nodes,~
9659 extra-left-margin,~
9660 extra-right-margin,~
9661 first-col,~
9662 first-row,~
9663 hlines,~
9664 hvlines,~
9665 hvlines-except-borders,~
9666 label,~
9667 last-col,~
9668 last-row,~
9669 left-margin,~
9670 light-syntax,~
9671 name,~
9672 notes~(several~subkeys),~
9673 nullify-dots,~
9674 pgf-node-code,~
9675 renew-dots,~
9676 respect-arraystretch,~
9677 right-margin,~
9678 rounded-corners,~
9679 rules~(with~the~subkeys~'color'~and~'width'),~
9680 short-caption,~
9681 t,~
9682 tabularnote,~
9683 vlines,~
9684 xdots/color,~
9685 xdots/shorten-start,~
9686 xdots/shorten-end,~
9687 xdots/shorten~and~
9688 xdots/line-style.
9689 }
9690 \@@_msg_new:nnn { Duplicate-name }
9691 {
9692 Duplicate-name.\\
9693 The~name~'\l_keys_value_tl'~is~already~used~and~you~shouldn't~use~
9694 the~same~environment~name~twice.~You~can~go~on,~but,~
9695 maybe,~you~will~have~incorrect~results~especially~

```

```

9696   if~you~use~'columns~width=auto'~.~If~you~don't~want~to~see~this~
9697   message~again,~use~the~key~'allow~duplicate~names'~in~
9698   '\token_to_str:N \NiceMatrixOptions'.\\
9699   \bool_if:NF \g_@@_messages_for_Overleaf_bool
9700   { For~a~list~of~the~names~already~used,~type~H~<return>. }
9701   }
9702   {
9703   The~names~already~defined~in~this~document~are:~
9704   \seq_use:Nnnn \g_@@_names_seq { ~and~ } { ,~ } { ~and~ }.
9705   }
9706   \@@_msg_new:nn { Option~auto~for~columns~width }
9707   {
9708   Erroneous~use.\\
9709   You~can't~give~the~value~'auto'~to~the~key~'columns~width'~here.~
9710   That~key~will~be~ignored.
9711   }

```

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	23
8	The options	24
9	Important code used by <code>{NiceArrayWithDelims}</code>	35
10	The <code>\CodeBefore</code>	47
11	The environment <code>{NiceArrayWithDelims}</code>	51
12	We construct the preamble of the array	56
13	The redefinition of <code>\multicolumn</code>	71
14	The environment <code>{NiceMatrix}</code> and its variants	88
15	<code>{NiceTabular}</code> , <code>{NiceTabularX}</code> and <code>{NiceTabular*}</code>	89
16	After the construction of the array	90
17	We draw the dotted lines	97
18	The actual instructions for drawing the dotted lines with Tikz	110
19	User commands available in the new environments	115
20	The command <code>\line</code> accessible in code-after	121
21	The command <code>\RowStyle</code>	123
22	Colors of cells, rows and columns	125
23	The vertical and horizontal rules	134
24	The key corners	149
25	The environment <code>{NiceMatrixBlock}</code>	152
26	The extra nodes	153
27	The blocks	157
28	How to draw the dotted lines transparently	176
29	Automatic arrays	177
30	The redefinition of the command <code>\dotfill</code>	178

31	The command <code>\diagbox</code>	179
32	The keyword <code>\CodeAfter</code>	180
33	The delimiters in the preamble	181
34	The command <code>\SubMatrix</code>	182
35	Les commandes <code>\UnderBrace</code> et <code>\OverBrace</code>	191
36	The command <code>\ShowCellNames</code>	194
37	We process the options at package loading	196
38	About the package underscore	198
39	Error messages of the package	198