

The code of the package `nicematrix`*

F. Pantigny
fpantigny@wanadoo.fr

September 2, 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 registred 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 6.23 of `nicematrix`, at the date of 2023/09/02.

```

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.
Be careful: that command is *not* “fully expandable” (because of `\\peek_meaning:NTF`).

```

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

```

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

```

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

```

4 Technical definitions

The following 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_pgful@check@rerun \pgfutil@check@rerun

```

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

```

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_pgful@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:nn { 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_pgfpointanchor \pgfpointanchor

```

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 raised when 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 content of that token list will be written on the `aux` file at the end of the environment (in an instruction `\tl_gset:cn { c_@@_ \int_use:N \g_@@_env_int _ tl }`).

```
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) `\l_@@_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 `\tablarnote`

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

- The caption may of course be provided by the command `\caption` in a floating environment. Of course, a command `\tablarnote` in that `\caption` makes sens only if the `\caption` is *before* the `{tabular}`.
- It’s also possible to use `\tablarnote` in the value of the key `caption` of the `{NiceTabular}` when the key `caption-above` is in force. However, in that case, one must remind that the caption is composed *after* the composition of the box which contains the main tabular (that’s mandatory since that caption must be wrapped with a line width equal to the width of the tabular). However, we want the labels of the successive tabular notes in the logical order. That’s why:

- The number of tabular notes present in the caption will be written on the `aux` file and available in `\g_@@_notes_caption_int`.³
- During the composition of the main tabular, the tabular notes will be numbered from `\g_@@_notes_caption_int+1` and the notes will be stored in `\g_@@_notes_seq`. Each composant of `\g_@@_notes_seq` will be a kind of couple of the form : `{label}{text of the tablarnote}`. The first composante is the optional argument (between square brackets) of the command `\tablarnote` (if the optional argument is not used, the value will be the special marker `\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 `tablarnote` will be used to count the tabular notes during the construction of the array (this counter won’t be used during the composition of the notes at the end of the array). You use a LaTeX counter because we will use `\refstepcounter` in order to have the tabular notes referenceable.

```

501 \newcounter { tablarnote }
```

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

```

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 \@captive || \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:nNf \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 \wedge , $_$ and \cdot . We use \tl_put_right:Nn and not \tl_set:Nn (or \tl_set:N) because we don't want a direct use of up= ... erased by a absent $\wedge\{\dots\}$.

```

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 $\backslash\text{Ddots}$ and $\backslash\text{Iddots}$, will be caught when $\backslash\text{Ddots}$ or $\backslash\text{Iddots}$ is used (during the construction of the array and not when we draw the dotted lines).

```

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:n { 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 `arydshln`, 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     }
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 } }

```

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

```

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@tablarnote { \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 O { } m ! O { } t \CodeBefore }
1817 {
1818     \bool_if:NT \c_@@_revtex_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 exposant 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_gzero: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 carateristics of the blocks (specified by `\Block`) of the array. The sequence `\g_@@_pos_of_blocks_seq` will contain only the position of the blocks (except the blocks with the key `hvlines`).

```
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 exposant 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 {

```

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

```

2057 \bool_lazy_or:nnT
2058 { ! \str_if_eq_p:Vn \g_@@_left_delim_tl { . } }
2059 { ! \str_if_eq_p:Vn \g_@@_right_delim_tl { . } }
2060 { \bool_gset_true:N \g_@@_delims_bool }

```

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

```

2061 \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).

```

2062 \int_compare:nNnTF \l_@@_first_col_int = 0
2063 { \tl_gput_left:NV \g_@@_preamble_tl \c_@@_preamble_first_col_tl }
2064 {
2065   \bool_lazy_all:nT
2066   {
2067     { \bool_not_p:n \g_@@_delims_bool }
2068     { \bool_not_p:n \l_@@_tabular_bool }
2069     { \tl_if_empty_p:N \l_@@_vlines_clist }
2070     { \bool_not_p:n \l_@@_exterior_arraycolsep_bool }
2071   }
2072   { \tl_gput_left:Nn \g_@@_preamble_tl { @ { } } }
2073 }
2074 \int_compare:nNnTF \l_@@_last_col_int > { -1 }
2075 { \tl_gput_right:NV \g_@@_preamble_tl \c_@@_preamble_last_col_tl }
2076 {
2077   \bool_lazy_all:nT
2078   {
2079     { \bool_not_p:n \g_@@_delims_bool }
2080     { \bool_not_p:n \l_@@_tabular_bool }
2081     { \tl_if_empty_p:N \l_@@_vlines_clist }
2082     { \bool_not_p:n \l_@@_exterior_arraycolsep_bool }
2083   }
2084   { \tl_gput_right:Nn \g_@@_preamble_tl { @ { } } }
2085 }

```

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`).

```

2086 \dim_compare:nNnT \l_@@_tabular_width_dim = \c_zero_dim
2087 {
2088   \tl_gput_right:Nn \g_@@_preamble_tl
2089   { > { \@@_error_too_much_cols: } 1 }
2090 }
2091 }

```

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

```

2092 \cs_new_protected:Npn \@@_patch_preamble:n #1
2093 {
2094   \str_case:nnF { #1 }
2095   {
2096     c { \@@_patch_preamble_i:n #1 }
2097     l { \@@_patch_preamble_i:n #1 }
2098     r { \@@_patch_preamble_i:n #1 }
2099     > { \@@_patch_preamble_xiv:n }
2100     ! { \@@_patch_preamble_ii:nn #1 }
2101     @ { \@@_patch_preamble_ii:nn #1 }
2102     | { \@@_patch_preamble_iii:n #1 }
2103     p { \@@_patch_preamble_iv:n #1 }
2104     b { \@@_patch_preamble_iv:n #1 }
2105     m { \@@_patch_preamble_iv:n #1 }
2106     \@@_V: { \@@_patch_preamble_v:n }
2107     V { \@@_patch_preamble_v:n }
2108     \@@_w: { \@@_patch_preamble_vi:nnnn { } #1 }
2109     \@@_W: { \@@_patch_preamble_vi:nnnn { \@@_special_W: } #1 }
2110     \@@_S: { \@@_patch_preamble_vii:n }
2111     ( { \@@_patch_preamble_viii:nn #1 }
2112     [ { \@@_patch_preamble_viii:nn #1 }
2113     \{ { \@@_patch_preamble_viii:nn #1 }
2114     \left { \@@_patch_preamble_viii:nn }
2115     ) { \@@_patch_preamble_ix:nn #1 }
2116     ] { \@@_patch_preamble_ix:nn #1 }

```

```

2117     \}      { \@@_patch_preamble_ix:nn #1 }
2118     \right { \@@_patch_preamble_ix:nn }
2119     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}`.

```

2120     \@@_X { \@@_patch_preamble_x:n }
2121     \q_stop { }
2122 }
2123 {
2124     \str_if_eq:nVTF { #1 } \l_@@_letter_vlism_tl
2125     {
2126         \seq_gput_right:Nx \g_@@_cols_vlism_seq
2127         { \int_eval:n { \c@jCol + 1 } }
2128         \tl_gput_right:Nx \g_@@_preamble_tl
2129         { \exp_not:N ! { \skip_horizontal:N \arrayrulewidth } }
2130         \@@_patch_preamble:n
2131     }

```

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).

```

2132     {
2133         \keys_if_exist:nnTF { NiceMatrix / ColumnTypes } { #1 }
2134         {
2135             \keys_set:nn { NiceMatrix / ColumnTypes } { #1 }
2136             \@@_patch_preamble:n
2137         }
2138         {
2139             \tl_if_eq:nnT { #1 } { S }
2140             { \@@_fatal:n { unknown~column~type~S } }
2141             { \@@_fatal:nn { unknown~column~type } { #1 } }
2142         }
2143     }
2144 }
2145 }

```

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`

```

2146 \cs_new_protected:Npn \@@_patch_preamble_i:n #1
2147 {
2148     \tl_gput_right:NV \g_@@_preamble_tl \g_@@_pre_cell_tl
2149     \tl_gclear:N \g_@@_pre_cell_tl
2150     \tl_gput_right:Nn \g_@@_preamble_tl
2151     {
2152         > { \@@_cell_begin:w \str_set:Nn \l_@@_hpos_cell_str { #1 } }
2153         #1
2154         < \@@_cell_end:
2155     }

```

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

```

2156     \int_gincr:N \c@jCol
2157     \@@_patch_preamble_xi:n
2158 }

```

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

```

2159 \cs_new_protected:Npn \@@_patch_preamble_ii:nn #1 #2
2160 {

```

```

2161 \tl_gput_right:Nn \g_@@_preamble_tl { #1 { #2 } }
2162 \@@_patch_preamble:n
2163 }

```

For |

```

2164 \cs_new_protected:Npn \@@_patch_preamble_iii:n #1
2165 {
\l_tmpa_int is the number of successive occurrences of |
2166 \int_incr:N \l_tmpa_int
2167 \@@_patch_preamble_iii_i:n
2168 }

2169 \cs_new_protected:Npn \@@_patch_preamble_iii_i:n #1
2170 {
2171 \str_if_eq:nnTF { #1 } |
2172 { \@@_patch_preamble_iii:n | }
2173 {
2174 \str_if_eq:nnTF { #1 } [
2175 { \@@_patch_preamble_iii_ii:nw { } [ ] }
2176 { \@@_patch_preamble_iii_ii:nw { #1 } [ ] ] }
2177 }
2178 }

2179 \cs_new_protected:Npn \@@_patch_preamble_iii_ii:nw #1 [ #2 ]
2180 {
2181 \@@_compute_rule_width:n { multiplicity = \l_tmpa_int , #2 }
2182 \tl_gput_right:Nx \g_@@_preamble_tl
2183 {

```

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

```

2184 \exp_not:N ! { \skip_horizontal:n { \dim_eval:n { \l_@@_rule_width_dim } } }
2185 }
2186 \tl_gput_right:Nx \g_@@_pre_code_after_tl
2187 {
2188 \@@_vline:n
2189 {
2190 position = \int_eval:n { \c@jCol + 1 } ,
2191 multiplicity = \int_use:N \l_tmpa_int ,
2192 total-width = \dim_use:N \l_@@_rule_width_dim ,
2193 #2
2194 }

```

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.

```

2195 }
2196 \int_zero:N \l_tmpa_int
2197 \str_if_eq:nnT { #1 } { \q_stop } { \bool_gset_true:N \g_tmpb_bool }
2198 \@@_patch_preamble:n #1
2199 }

2200 \cs_new_protected:Npn \@@_patch_preamble_xiv:n #1
2201 {
2202 \tl_gput_right:Nn \g_@@_pre_cell_tl { > { #1 } }
2203 \@@_patch_preamble:n
2204 }

2205 \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.

```

2206 \keys_define:nn { WithArrows / p-column }
2207 {
2208 r .code:n = \str_set:Nn \l_@@_hpos_col_str { r } ,
2209 r .value_forbidden:n = true ,
2210 c .code:n = \str_set:Nn \l_@@_hpos_col_str { c } ,

```

```

2211 c .value_forbidden:n = true ,
2212 l .code:n = \str_set:Nn \l_@@_hpos_col_str { l } ,
2213 l .value_forbidden:n = true ,
2214 R .code:n =
2215   \IfPackageLoadedTF { ragged2e }
2216   { \str_set:Nn \l_@@_hpos_col_str { R } }
2217   {
2218     \@@_error_or_warning:n { ragged2e-not-loaded }
2219     \str_set:Nn \l_@@_hpos_col_str { r }
2220   } ,
2221 R .value_forbidden:n = true ,
2222 L .code:n =
2223   \IfPackageLoadedTF { ragged2e }
2224   { \str_set:Nn \l_@@_hpos_col_str { L } }
2225   {
2226     \@@_error_or_warning:n { ragged2e-not-loaded }
2227     \str_set:Nn \l_@@_hpos_col_str { l }
2228   } ,
2229 L .value_forbidden:n = true ,
2230 C .code:n =
2231   \IfPackageLoadedTF { ragged2e }
2232   { \str_set:Nn \l_@@_hpos_col_str { C } }
2233   {
2234     \@@_error_or_warning:n { ragged2e-not-loaded }
2235     \str_set:Nn \l_@@_hpos_col_str { c }
2236   } ,
2237 C .value_forbidden:n = true ,
2238 S .code:n = \str_set:Nn \l_@@_hpos_col_str { si } ,
2239 S .value_forbidden:n = true ,
2240 p .code:n = \str_set:Nn \l_@@_vpos_col_str { p } ,
2241 p .value_forbidden:n = true ,
2242 t .meta:n = p ,
2243 m .code:n = \str_set:Nn \l_@@_vpos_col_str { m } ,
2244 m .value_forbidden:n = true ,
2245 b .code:n = \str_set:Nn \l_@@_vpos_col_str { b } ,
2246 b .value_forbidden:n = true ,
2247 }

```

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

```

2248 \cs_new_protected:Npn \@@_patch_preamble_iv:n #1
2249 {
2250   \str_set:Nn \l_@@_vpos_col_str { #1 }

```

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

```

2251   \@@_patch_preamble_iv_i:n
2252 }

2253 \cs_new_protected:Npn \@@_patch_preamble_iv_i:n #1
2254 {
2255   \str_if_eq:nnTF { #1 } { [ ]
2256     { \@@_patch_preamble_iv_ii:w [ ]
2257       { \@@_patch_preamble_iv_ii:w [ ] { #1 } }
2258     }

2259   \cs_new_protected:Npn \@@_patch_preamble_iv_ii:w [ #1 ]
2260   { \@@_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.

```

2261 \cs_new_protected:Npn \@@_patch_preamble_iv_iii:nn #1 #2
2262 {

```

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).

```

2263   \str_set:Nn \l_@@_hpos_col_str { j }

```

```

2264 \tl_set:Nn \l_tmpa_tl { #1 }
2265 \tl_replace_all:Nnn \l_tmpa_tl { \@@_S: } { S }
2266 \@@_keys_p_column:V \l_tmpa_tl
2267 \@@_patch_preamble_iv_iv:nn { #2 } { minipage }
2268 }
2269 \cs_new_protected:Npn \@@_keys_p_column:n #1
2270 { \keys_set_known:nnN { WithArrows / p-column } { #1 } \l_tmpa_tl }
2271 \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`.

```

2272 \cs_new_protected:Npn \@@_patch_preamble_iv_iv:nn #1 #2
2273 {
2274   \use:x
2275   {
2276     \@@_patch_preamble_iv_v:nnnnnnnn
2277     { \str_if_eq:VnTF \l_@@_vpos_col_str { p } { t } { b } }
2278     { \dim_eval:n { #1 } }
2279   }

```

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.

```

2280   \str_if_eq:VnTF \l_@@_hpos_col_str j
2281   { \str_set:Nn \exp_not:N \l_@@_hpos_cell_str { } }
2282   {
2283     \str_set:Nn \exp_not:N \l_@@_hpos_cell_str
2284     { \str_lowercase:V \l_@@_hpos_col_str }
2285   }
2286   \str_case:Vn \l_@@_hpos_col_str
2287   {
2288     c { \exp_not:N \centering }
2289     l { \exp_not:N \raggedright }
2290     r { \exp_not:N \raggedleft }
2291     C { \exp_not:N \Centering }
2292     L { \exp_not:N \RaggedRight }
2293     R { \exp_not:N \RaggedLeft }
2294   }
2295 }
2296 { \str_if_eq:VnT \l_@@_vpos_col_str { m } \@@_center_cell_box: }
2297 { \str_if_eq:VnT \l_@@_hpos_col_str { si } \siunitx_cell_begin:w }
2298 { \str_if_eq:VnT \l_@@_hpos_col_str { si } \siunitx_cell_end: }
2299 { #2 }
2300 {
2301   \str_case:VnF \l_@@_hpos_col_str
2302   {
2303     { j } { c }
2304     { si } { c }
2305   }

```

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

```

2306   { \str_lowercase:V \l_@@_hpos_col_str }
2307 }
2308 }

```

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

```

2309 \int_gincr:N \c@jCol
2310 \@@_patch_preamble_xi:n
2311 }

```

`#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*.

```

2312 \cs_new_protected:Npn \@@_patch_preamble_iv_v:nnnnnnnn #1 #2 #3 #4 #5 #6 #7 #8
2313 {
2314   \str_if_eq:VnTF \l_@@_hpos_col_str { si }
2315     { \tl_gput_right:Nn \g_@@_preamble_tl { > { \@@_test_if_empty_for_S: } } }
2316     { \tl_gput_right:Nn \g_@@_preamble_tl { > { \@@_test_if_empty: } } }
2317   \tl_gput_right:NV \g_@@_preamble_tl \g_@@_pre_cell_tl
2318   \tl_gclear:N \g_@@_pre_cell_tl
2319   \tl_gput_right:Nn \g_@@_preamble_tl
2320     {
2321       > {

```

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.

```

2322   \dim_set:Nn \l_@@_col_width_dim { #2 }
2323   \@@_cell_begin:w
2324   \begin { #7 } [ #1 ] { #2 }

```

The following lines have been taken from array.sty.

```

2325   \everypar
2326   {
2327     \vrule height \box_ht:N \@arstrutbox width \c_zero_dim
2328     \everypar { }
2329   }

```

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

```

2330   #3

```

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

```

2331   \g_@@_row_style_tl
2332   \arraybackslash
2333   #5
2334   }
2335   #8
2336   < {
2337   #6

```

The following line has been taken from array.sty.

```

2338   \@finalstrut \@arstrutbox
2339   % \bool_if:NT \g_@@_rotate_bool { \raggedright \hsize = 3 cm }
2340   \end { #7 }

```

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

```

2341   #4
2342   \@@_cell_end:
2343   }
2344   }
2345   }

```

```

2346 \cs_new_protected:Npn \@@_test_if_empty: \ignorespaces #1
2347 {
2348   \peek_meaning:NT \unskip
2349   {
2350     \tl_gput_right:Nn \g_@@_cell_after_hook_tl

```

```

2351     {
2352         \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.

```

2353         \skip_horizontal:N \l_@@_col_width_dim
2354     }
2355 }
2356 #1
2357 }
2358 \cs_new_protected:Npn \@@_test_if_empty_for_S: #1
2359 {
2360     \peek_meaning:NT \_siunitx_table_skip:n
2361     {
2362         \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2363         { \box_set_wd:Nn \l_@@_cell_box \c_zero_dim }
2364     }
2365     #1
2366 }

```

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.

```

2367 \cs_new_protected:Npn \@@_center_cell_box:
2368 {

```

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

```

2369     \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2370     {
2371         \int_compare:nNnT
2372         { \box_ht:N \l_@@_cell_box }
2373         >

```

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).

```

2374         { \box_ht:N \strutbox }
2375     {
2376         \hbox_set:Nn \l_@@_cell_box
2377         {
2378             \box_move_down:nn
2379             {
2380                 ( \box_ht:N \l_@@_cell_box - \box_ht:N \@arstrutbox
2381                 + \baselineskip ) / 2
2382             }
2383             { \box_use:N \l_@@_cell_box }
2384         }
2385     }
2386 }
2387 }

```

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

```

2388 \cs_new_protected:Npn \@@_patch_preamble_v:n #1
2389 {
2390     \str_if_eq:nnTF { #1 } { [ ] }
2391     { \@@_patch_preamble_v_i:w [ ] }
2392     { \@@_patch_preamble_v_i:w [ ] { #1 } }
2393 }
2394 \cs_new_protected:Npn \@@_patch_preamble_v_i:w [ #1 ]

```

```

2395 { \@@_patch_preamble_v_ii:nn { #1 } }
2396 \cs_new_protected:Npn \@@_patch_preamble_v_ii:nn #1 #2
2397 {
2398   \str_set:Nn \l_@@_vpos_col_str { p }
2399   \str_set:Nn \l_@@_hpos_col_str { j }
2400   \tl_set:Nn \l_tmpa_tl { #1 }
2401   \tl_replace_all:Nnn \l_tmpa_tl { \@@_S: } { S }
2402   \@@_keys_p_column:V \l_tmpa_tl
2403   \IfPackageLoadedTF { varwidth }
2404     { \@@_patch_preamble_iv_iv:nn { #2 } { varwidth } }
2405     {
2406       \@@_error_or_warning:n { varwidth~not~loaded }
2407       \@@_patch_preamble_iv_iv:nn { #2 } { minipage }
2408     }
2409 }

```

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.

```

2410 \cs_new_protected:Npn \@@_patch_preamble_vi:nnnn #1 #2 #3 #4
2411 {
2412   \str_if_eq:nnTF { #3 } { s }
2413     { \@@_patch_preamble_vi_i:nnnn { #1 } { #4 } }
2414     { \@@_patch_preamble_vi_ii:nnnn { #1 } { #2 } { #3 } { #4 } }
2415 }

```

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.

```

2416 \cs_new_protected:Npn \@@_patch_preamble_vi_i:nnnn #1 #2
2417 {
2418   \tl_gput_right:NV \g_@@_preamble_tl \g_@@_pre_cell_tl
2419   \tl_gclear:N \g_@@_pre_cell_tl
2420   \tl_gput_right:Nn \g_@@_preamble_tl
2421     {
2422       > {
2423         \dim_set:Nn \l_@@_col_width_dim { #2 }
2424         \@@_cell_begin:w
2425         \str_set:Nn \l_@@_hpos_cell_str { c }
2426       }
2427       c
2428       < {
2429         \@@_cell_end_for_w_s:
2430         #1
2431         \@@_adjust_size_box:
2432         \box_use_drop:N \l_@@_cell_box
2433       }
2434     }
2435   \int_gincr:N \c@jCol
2436   \@@_patch_preamble_xi:n
2437 }

```

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

```

2438 \cs_new_protected:Npn \@@_patch_preamble_vi_ii:nnnn #1 #2 #3 #4
2439 {
2440   \tl_gput_right:NV \g_@@_preamble_tl \g_@@_pre_cell_tl
2441   \tl_gclear:N \g_@@_pre_cell_tl
2442   \tl_gput_right:Nn \g_@@_preamble_tl
2443     {
2444       > {

```

The parameter `\l_@@_col_width_dim`, which is the width of the current column, will be available in each cell of the column. It will be used by the mono-column blocks.

```

2445         \dim_set:Nn \l_@@_col_width_dim { #4 }
2446         \hbox_set:Nw \l_@@_cell_box
2447         \@@_cell_begin:w
2448         \str_set:Nn \l_@@_hpos_cell_str { #3 }
2449     }
2450     c
2451     < {
2452         \@@_cell_end:
2453         \hbox_set_end:
2454         #1
2455         \@@_adjust_size_box:
2456         \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2457     }
2458 }

```

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

```

2459     \int_gincr:N \c@jCol
2460     \@@_patch_preamble_xi:n
2461 }

```

```

2462 \cs_new_protected:Npn \@@_special_W:
2463 {
2464     \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \l_@@_col_width_dim
2465     { \@@_warning:n { W~warning } }
2466 }

```

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).

```

2467 \cs_new_protected:Npn \@@_patch_preamble_vii:n #1
2468 {
2469     \str_if_eq:nnTF { #1 } { [ ]
2470     { \@@_patch_preamble_vii_i:w [ ]
2471     { \@@_patch_preamble_vii_i:w [ ] { #1 } }
2472     }
2473     \cs_new_protected:Npn \@@_patch_preamble_vii_i:w [ #1 ]
2474     { \@@_patch_preamble_vii_ii:n { #1 } }
2475     \cs_new_protected:Npn \@@_patch_preamble_vii_ii:n #1
2476     {
2477         \IfPackageAtLeastTF { siunitx } { 2022/01/01 }
2478         {
2479             \tl_gput_right:NV \g_@@_preamble_tl \g_@@_pre_cell_tl
2480             \tl_gclear:N \g_@@_pre_cell_tl
2481             \tl_gput_right:Nn \g_@@_preamble_tl
2482             {
2483                 > {
2484                     \@@_cell_begin:w
2485                     \keys_set:nn { siunitx } { #1 }
2486                     \siunitx_cell_begin:w
2487                 }
2488                 c
2489                 < { \siunitx_cell_end: \@@_cell_end: }
2490             }

```

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

```

2491     \int_gincr:N \c@jCol
2492     \@@_patch_preamble_xi:n
2493 }
2494 { \@@_fatal:n { Version~of~siunitx~too~old } }
2495 }

```

For (, [and \{.

```

2496 \cs_new_protected:Npn \@@_patch_preamble_viii:nn #1 #2
2497 {
2498   \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.

```

2499   \int_compare:nNnTF \c@jCol = \c_zero_int
2500   {
2501     \str_if_eq:VnTF \g_@@_left_delim_tl { . }
2502     {

```

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

```

2503       \tl_gset:Nn \g_@@_left_delim_tl { #1 }
2504       \tl_gset:Nn \g_@@_right_delim_tl { . }
2505       \@@_patch_preamble:n #2
2506     }
2507   {
2508     \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } }
2509     \@@_patch_preamble_viii_i:nn { #1 } { #2 }
2510   }
2511 }
2512 { \@@_patch_preamble_viii_i:nn { #1 } { #2 } }
2513 }
2514 \cs_new_protected:Npn \@@_patch_preamble_viii_i:nn #1 #2
2515 {
2516   \tl_gput_right:Nx \g_@@_pre_code_after_tl
2517   { \@@_delimiter:nnn #1 { \int_eval:n { \c@jCol + 1 } } } \c_true_bool }
2518   \tl_if_in:nnTF { ( [ \{ ) ] \} } \left \right { #2 }
2519   {
2520     \@@_error:nn { delimiter~after~opening } { #2 }
2521     \@@_patch_preamble:n
2522   }
2523   { \@@_patch_preamble:n #2 }
2524 }

```

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}).

```

2525 \cs_new_protected:Npn \@@_patch_preamble_ix:nn #1 #2
2526 {
2527   \bool_if:NT \l_@@_small_bool { \@@_fatal:n { Delimiter~with~small } }
2528   \tl_if_in:nnTF { ) ] \} } { #2 }
2529   { \@@_patch_preamble_ix_i:nnn #1 #2 }
2530   {
2531     \tl_if_eq:nnTF { \q_stop } { #2 }
2532     {
2533       \str_if_eq:VnTF \g_@@_right_delim_tl { . }
2534       { \tl_gset:Nn \g_@@_right_delim_tl { #1 } }
2535       {
2536         \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } }
2537         \tl_gput_right:Nx \g_@@_pre_code_after_tl
2538         { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } } \c_false_bool }
2539       \@@_patch_preamble:n #2
2540     }
2541   }
2542   {
2543     \tl_if_in:nnT { ( [ \{ \left } { #2 }
2544     { \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } } }
2545     \tl_gput_right:Nx \g_@@_pre_code_after_tl
2546     { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } } \c_false_bool }
2547     \@@_patch_preamble:n #2

```

```

2548     }
2549   }
2550 }
2551 \cs_new_protected:Npn \@@_patch_preamble_ix_i:nnn #1 #2 #3
2552 {
2553   \tl_if_eq:nnTF { \q_stop } { #3 }
2554   {
2555     \str_if_eq:VnTF \g_@@_right_delim_tl { . }
2556     {
2557       \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } }
2558       \tl_gput_right:Nx \g_@@_pre_code_after_tl
2559       { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2560       \tl_gset:Nn \g_@@_right_delim_tl { #2 }
2561     }
2562     {
2563       \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } }
2564       \tl_gput_right:Nx \g_@@_pre_code_after_tl
2565       { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2566       \@@_error:nn { double~closing~delimiter } { #2 }
2567     }
2568   }
2569   {
2570     \tl_gput_right:Nx \g_@@_pre_code_after_tl
2571     { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2572     \@@_error:nn { double~closing~delimiter } { #2 }
2573     \@@_patch_preamble:n #3
2574   }
2575 }

```

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.

```

2576 \cs_new_protected:Npn \@@_patch_preamble_x:n #1
2577 {
2578   \str_if_eq:nnTF { #1 } { [ ]
2579   { \@@_patch_preamble_x_i:w [ ]
2580   { \@@_patch_preamble_x_i:w [ ] #1 }
2581 }
2582 \cs_new_protected:Npn \@@_patch_preamble_x_i:w [ #1 ]
2583 { \@@_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).

```

2584 \keys_define:nn { WithArrows / X-column }
2585 { 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.

```

2586 \cs_new_protected:Npn \@@_patch_preamble_x_ii:n #1
2587 {

```

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).

```

2588   \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).

```

2589   \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`.

```

2590 \int_zero_new:N \l_@@_weight_int
2591 \int_set:Nn \l_@@_weight_int { 1 }
2592 \tl_set:Nn \l_tmpa_tl { #1 }
2593 \tl_replace_all:Nnn \l_tmpa_tl { \@@_S: } { S }
2594 \@@_keys_p_column:V \l_tmpa_tl
2595 \keys_set:nV { WithArrows / X-column } \l_tmpa_tl
2596 \int_compare:nNnT \l_@@_weight_int < 0
2597 {
2598   \@@_error_or_warning:n { negative~weight }
2599   \int_set:Nn \l_@@_weight_int { - \l_@@_weight_int }
2600 }
2601 \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).

```

2602 \bool_if:NTF \l_@@_X_columns_aux_bool
2603 {
2604   \exp_args:Nnx
2605   \@@_patch_preamble_iv_iv:nn
2606   { \l_@@_weight_int \l_@@_X_columns_dim }
2607   { minipage }
2608 }
2609 {
2610   \tl_gput_right:Nn \g_@@_preamble_tl
2611   {
2612     > {
2613       \@@_cell_begin:w
2614       \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`.

```

2615 \NotEmpty

```

The following code will nullify the box of the cell.

```

2616 \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2617 { \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`.

```

2618 \begin { minipage } { 5 cm } \arraybackslash
2619 }
2620 c
2621 < {
2622   \end { minipage }
2623   \@@_cell_end:
2624 }
2625 }
2626 \int_gincr:N \c@jCol
2627 \@@_patch_preamble_xi:n
2628 }
2629 }

```

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 `@{...}`.

```

2630 \cs_new_protected:Npn \@@_patch_preamble_xi:n #1
2631 {

```

```

2632 \str_if_eq:nnTF { #1 } { < }
2633 \@@_patch_preamble_xiii:n
2634 {
2635   \str_if_eq:nnTF { #1 } { @ }
2636   \@@_patch_preamble_xv:n
2637   {
2638     \tl_if_eq:NnTF \l_@@_vlines_clist { all }
2639     {
2640       \tl_gput_right:Nn \g_@@_preamble_tl
2641       { ! { \skip_horizontal:N \arrayrulewidth } }
2642     }
2643     {
2644       \exp_args:NNx
2645       \clist_if_in:NnT \l_@@_vlines_clist { \int_eval:n { \c@jCol + 1 } }
2646       {
2647         \tl_gput_right:Nn \g_@@_preamble_tl
2648         { ! { \skip_horizontal:N \arrayrulewidth } }
2649       }
2650     }
2651     \@@_patch_preamble:n { #1 }
2652   }
2653 }
2654 }
2655 \cs_new_protected:Npn \@@_patch_preamble_xiii:n #1
2656 {
2657   \tl_gput_right:Nn \g_@@_preamble_tl { < { #1 } }
2658   \@@_patch_preamble_xi:n
2659 }

```

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.

```

2660 \cs_new_protected:Npn \@@_patch_preamble_xv:n #1
2661 {
2662   \tl_if_eq:NnTF \l_@@_vlines_clist { all }
2663   {
2664     \tl_gput_right:Nn \g_@@_preamble_tl
2665     { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2666   }
2667   {
2668     \exp_args:NNx
2669     \clist_if_in:NnTF \l_@@_vlines_clist { \int_eval:n { \c@jCol + 1 } }
2670     {
2671       \tl_gput_right:Nn \g_@@_preamble_tl
2672       { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2673     }
2674     { \tl_gput_right:Nn \g_@@_preamble_tl { @ { #1 } } }
2675   }
2676   \@@_patch_preamble:n
2677 }
2678 \cs_new_protected:Npn \@@_set_preamble:Nn #1 #2
2679 {
2680   \group_begin:
2681   \@@_newcolumnntype w [ 2 ] { \@@_w: { ##1 } { ##2 } }
2682   \@@_newcolumnntype W [ 2 ] { \@@_W: { ##1 } { ##2 } }
2683   \@temptokena { #2 }
2684   \@tempswatrue
2685   \@whilesw \if@tempswa \fi { \@tempswafalse \the \NC@list }
2686   \tl_gclear:N \g_@@_preamble_tl
2687   \exp_after:wN \@@_patch_m_preamble:n \the \@temptokena \q_stop
2688   \group_end:
2689   \tl_set_eq:NN #1 \g_@@_preamble_tl
2690 }

```

13 The redefinition of `\multicolumn`

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

```
2691 \cs_new:Npn \@@_multicolumn:nnn #1 #2 #3
2692 {
```

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`.

```
2693 \multispan { #1 }
2694 \begingroup
2695 \cs_set:Npn \@addamp { \if@firstamp \@firstampfalse \else \@preamerr 5 \fi }
2696 \@@_newcolumnntype w [ 2 ] { \@@_w: { ##1 } { ##2 } }
2697 \@@_newcolumnntype W [ 2 ] { \@@_W: { ##1 } { ##2 } }
```

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

```
2698 \@temptokena = { #2 }
2699 \@tempswatrue
2700 \@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`.

```
2701 \tl_gclear:N \g_@@_preamble_tl
2702 \exp_after:wN \@@_patch_m_preamble:n \the \@temptokena \q_stop
```

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

```
2703 \exp_args:NV \mkpream \g_@@_preamble_tl
2704 \addtopreamble \empty
2705 \endgroup
```

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

```
2706 \int_compare:nNnT { #1 } > 1
2707 {
2708   \seq_gput_left:Nx \g_@@_multicolumn_cells_seq
2709   { \int_use:N \c@iRow - \int_eval:n { \c@jCol + 1 } }
2710   \seq_gput_left:Nn \g_@@_multicolumn_sizes_seq { #1 }
2711   \seq_gput_right:Nx \g_@@_pos_of_blocks_seq
2712   {
2713     {
2714       \int_compare:nNnTF \c@jCol = 0
2715       { \int_eval:n { \c@iRow + 1 } }
2716       { \int_use:N \c@iRow }
2717     }
2718     { \int_eval:n { \c@jCol + 1 } }
2719     {
2720       \int_compare:nNnTF \c@jCol = 0
2721       { \int_eval:n { \c@iRow + 1 } }
2722       { \int_use:N \c@iRow }
2723     }
2724     { \int_eval:n { \c@jCol + #1 } }
2725     { } % for the name of the block
2726   }
2727 }
```

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

```
2728 \cs_set:Npn \@sharp { #3 }
2729 \@arstrut
2730 \@preamble
2731 \null
```

We add some lines.

```

2732 \int_gadd:Nn \c@jCol { #1 - 1 }
2733 \int_compare:nNnT \c@jCol > \g_@@_col_total_int
2734 { \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
2735 \ignorespaces
2736 }

```

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`.

```

2737 \cs_new_protected:Npn \@@_patch_m_preamble:n #1
2738 {
2739   \str_case:nnF { #1 }
2740   {
2741     c { \@@_patch_m_preamble_i:n #1 }
2742     l { \@@_patch_m_preamble_i:n #1 }
2743     r { \@@_patch_m_preamble_i:n #1 }
2744     > { \@@_patch_m_preamble_ii:nn #1 }
2745     ! { \@@_patch_m_preamble_ii:nn #1 }
2746     @ { \@@_patch_m_preamble_ii:nn #1 }
2747     | { \@@_patch_m_preamble_iii:n #1 }
2748     p { \@@_patch_m_preamble_iv:nnn t #1 }
2749     m { \@@_patch_m_preamble_iv:nnn c #1 }
2750     b { \@@_patch_m_preamble_iv:nnn b #1 }
2751     \@@_w: { \@@_patch_m_preamble_v:nnnn { } #1 }
2752     \@@_W: { \@@_patch_m_preamble_v:nnnn { \@@_special_W: } #1 }
2753     \q_stop { }
2754   }
2755   {
2756     \tl_if_eq:nnT { #1 } { S }
2757     { \@@_fatal:n { unknown~column~type~S } }
2758     { \@@_fatal:nn { unknown~column~type } { #1 } }
2759   }
2760 }

```

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

```

2761 \cs_new_protected:Npn \@@_patch_m_preamble_i:n #1
2762 {
2763   \tl_gput_right:Nn \g_@@_preamble_tl
2764   {
2765     > { \@@_cell_begin:w \str_set:Nn \l_@@_hpos_cell_str { #1 } }
2766     #1
2767     < \@@_cell_end:
2768   }

```

We test for the presence of a `<`.

```

2769 \@@_patch_m_preamble_x:n
2770 }

```

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

```

2771 \cs_new_protected:Npn \@@_patch_m_preamble_ii:nn #1 #2
2772 {
2773   \tl_gput_right:Nn \g_@@_preamble_tl { #1 { #2 } }
2774   \@@_patch_m_preamble:n
2775 }

```

For `|`

```

2776 \cs_new_protected:Npn \@@_patch_m_preamble_iii:n #1
2777 {
2778   \tl_gput_right:Nn \g_@@_preamble_tl { #1 }
2779   \@@_patch_m_preamble:n
2780 }

```

For p, m and b

```

2781 \cs_new_protected:Npn \@@_patch_m_preamble_iv:nnn #1 #2 #3
2782 {
2783   \tl_gput_right:Nn \g_@@_preamble_tl
2784   {
2785     > {
2786       \@@_cell_begin:w
2787       \begin { minipage } [ #1 ] { \dim_eval:n { #3 } }
2788       \mode_leave_vertical:
2789       \arraybackslash
2790       \vrule height \box_ht:N \@@arstrutbox depth 0 pt width 0 pt
2791     }
2792     c
2793     < {
2794       \vrule height 0 pt depth \box_dp:N \@@arstrutbox width 0 pt
2795       \end { minipage }
2796       \@@_cell_end:
2797     }
2798   }

```

We test for the presence of a <.

```

2799   \@@_patch_m_preamble_x:n
2800 }

```

For w and W

```

2801 \cs_new_protected:Npn \@@_patch_m_preamble_v:nnnn #1 #2 #3 #4
2802 {
2803   \tl_gput_right:Nn \g_@@_preamble_tl
2804   {
2805     > {
2806       \dim_set:Nn \l_@@_col_width_dim { #4 }
2807       \hbox_set:Nw \l_@@_cell_box
2808       \@@_cell_begin:w
2809       \str_set:Nn \l_@@_hpos_cell_str { #3 }
2810     }
2811     c
2812     < {
2813       \@@_cell_end:
2814       \hbox_set_end:
2815       \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
2816       #1
2817       \@@_adjust_size_box:
2818       \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2819     }
2820   }

```

We test for the presence of a <.

```

2821   \@@_patch_m_preamble_x:n
2822 }

```

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

```

2823 \cs_new_protected:Npn \@@_patch_m_preamble_x:n #1
2824 {
2825   \str_if_eq:nnTF { #1 } { < }
2826   \@@_patch_m_preamble_ix:n
2827   { \@@_patch_m_preamble:n { #1 } }
2828 }
2829 \cs_new_protected:Npn \@@_patch_m_preamble_ix:n #1
2830 {
2831   \tl_gput_right:Nn \g_@@_preamble_tl { < { #1 } }
2832   \@@_patch_m_preamble_x:n
2833 }

```

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`).

```

2834 \cs_new_protected:Npn \@@_put_box_in_flow:
2835 {
2836   \box_set_ht:Nn \l_tmpa_box { \box_ht:N \l_tmpa_box + \l_tmpa_dim }
2837   \box_set_dp:Nn \l_tmpa_box { \box_dp:N \l_tmpa_box + \l_tmpb_dim }
2838   \tl_if_eq:NnTF \l_@@_baseline_tl { c }
2839     { \box_use_drop:N \l_tmpa_box }
2840     \@@_put_box_in_flow_i:
2841 }

```

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).

```

2842 \cs_new_protected:Npn \@@_put_box_in_flow_i:
2843 {
2844   \pgfpicture
2845     \@@_qpoint:n { row - 1 }
2846     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
2847     \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
2848     \dim_gadd:Nn \g_tmpa_dim \pgf@y
2849     \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).

```

2850   \str_if_in:NnTF \l_@@_baseline_tl { line- }
2851   {
2852     \int_set:Nn \l_tmpa_int
2853     {
2854       \str_range:Nnn
2855         \l_@@_baseline_tl
2856         6
2857         { \tl_count:V \l_@@_baseline_tl }
2858     }
2859     \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
2860   }
2861   {
2862     \str_case:VnF \l_@@_baseline_tl
2863     {
2864       { t } { \int_set:Nn \l_tmpa_int 1 }
2865       { b } { \int_set_eq:NN \l_tmpa_int \c@iRow }
2866     }
2867     { \int_set:Nn \l_tmpa_int \l_@@_baseline_tl }
2868     \bool_lazy_or:nnT
2869     { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
2870     { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
2871     {
2872       \@@_error:n { bad~value~for~baseline }
2873       \int_set:Nn \l_tmpa_int 1
2874     }
2875     \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }

```

We take into account the position of the mathematical axis.

```

2876     \dim_gsub:Nn \g_tmpa_dim { \fontdimen22 \textfont2 }
2877   }
2878   \dim_gsub:Nn \g_tmpa_dim \pgf@y

```

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

```

2879   \endpgfpicture
2880   \box_move_up:nn \g_tmpa_dim { \box_use_drop:N \l_tmpa_box }
2881   \box_use_drop:N \l_tmpa_box
2882 }

```

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).

```
2883 \cs_new_protected:Npn \@@_use_arraybox_with_notes_c:
2884 {
```

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.

```
2885 \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
2886 {
2887   \int_compare:nNnT \c@jCol > 1 % added 2023-08-13
2888   {
2889     \box_set_wd:Nn \l_@@_the_array_box
2890     { \box_wd:N \l_@@_the_array_box - \arraycolsep }
2891   }
2892 }
```

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

```
2893 \begin { minipage } [ t ] { \box_wd:N \l_@@_the_array_box }
2894 \bool_if:NT \l_@@_caption_above_bool
2895 {
2896   \tl_if_empty:NF \l_@@_caption_tl
2897   {
2898     \bool_set_false:N \g_@@_caption_finished_bool
2899     \int_gzero:N \c@tabularnote
2900     \@@_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).

```
2901     \int_compare:nNnT \g_@@_notes_caption_int > 0
2902     {
2903       \tl_gput_right:Nx \g_@@_aux_tl
2904       {
2905         \tl_set:Nn \exp_not:N \l_@@_note_in_caption_tl
2906         { \int_use:N \g_@@_notes_caption_int }
2907       }
2908       \int_gzero:N \g_@@_notes_caption_int
2909     }
2910   }
2911 }
```

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

```
2912 \hbox
2913 {
2914   \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.

```
2915 \@@_create_extra_nodes:
2916 \seq_if_empty:NF \g_@@_blocks_seq \@@_draw_blocks:
2917 }
```

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

```
2918 \bool_lazy_any:nT
2919 {
2920   { ! \seq_if_empty_p:N \g_@@_notes_seq }
```

```

2921     { ! \seq_if_empty_p:N \g_@@_notes_in_caption_seq }
2922     { ! \tl_if_empty_p:V \g_@@_tabularnote_tl }
2923   }
2924   \@@_insert_tabularnotes:
2925   \cs_set_eq:NN \tabularnote \@@_tabularnote_error:n
2926   \bool_if:NF \l_@@_caption_above_bool \@@_insert_caption:
2927   \end { minipage }
2928 }

```

```

2929 \cs_new_protected:Npn \@@_insert_caption:
2930 {
2931   \tl_if_empty:NF \l_@@_caption_tl
2932   {
2933     \cs_if_exist:NTF \@capttype
2934     { \@@_insert_caption_i: }
2935     { \@@_error:n { caption-outside~float } }
2936   }
2937 }

```

```

2938 \cs_new_protected:Npn \@@_insert_caption_i:
2939 {
2940   \group_begin:

```

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

```

2941   \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.

```

2942   \IfPackageLoadedTF { floatrow }
2943   { \cs_set_eq:NN \@makecaption \FR@makecaption }
2944   { }
2945   \tl_if_empty:NTF \l_@@_short_caption_tl
2946   { \caption }
2947   { \caption [ \l_@@_short_caption_tl ] }
2948   { \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.

```

2949   \bool_if:NF \g_@@_caption_finished_bool % added 2023/06/30
2950   {
2951     \bool_gset_true:N \g_@@_caption_finished_bool
2952     \int_gset_eq:NN \g_@@_notes_caption_int \c@tabularnote
2953     \int_gzero:N \c@tabularnote
2954   }
2955   \tl_if_empty:NF \l_@@_label_tl { \label { \l_@@_label_tl } }
2956   \group_end:
2957 }

```

```

2958 \cs_new_protected:Npn \@@_tabularnote_error:n #1
2959 {
2960   \@@_error_or_warning:n { tabularnote~below~the~tabular }
2961   \@@_gredirect_none:n { tabularnote~below~the~tabular }
2962 }
2963 \cs_new_protected:Npn \@@_insert_tabularnotes:
2964 {
2965   \seq_gconcat:NNN \g_@@_notes_seq \g_@@_notes_in_caption_seq \g_@@_notes_seq
2966   \int_set:Nn \c@tabularnote { \seq_count:N \g_@@_notes_seq }
2967   \skip_vertical:N 0.65ex

```

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

```

2968 \group_begin:
2969 \l_@@_notes_code_before_tl
2970 \tl_if_empty:NF \g_@@_tabularnote_tl
2971 {
2972   \g_@@_tabularnote_tl \par
2973   \tl_gclear:N \g_@@_tabularnote_tl
2974 }

```

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.

```

2975 \int_compare:nNnT \c@tabularnote > 0
2976 {
2977   \bool_if:NTF \l_@@_notes_para_bool
2978   {
2979     \begin { tabularnotes* }
2980     \seq_map_inline:Nn \g_@@_notes_seq
2981       { \@@_one_tabularnote:nn ##1 }
2982     \strut
2983     \end { tabularnotes* }

```

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

```

2984 \par
2985 }
2986 {
2987   \tabularnotes
2988   \seq_map_inline:Nn \g_@@_notes_seq
2989     { \@@_one_tabularnote:nn ##1 }
2990   \strut
2991   \endtabularnotes
2992 }
2993 }
2994 \unskip
2995 \group_end:
2996 \bool_if:NT \l_@@_notes_bottomrule_bool
2997 {
2998   \IfPackageLoadedTF { booktabs }
2999   {

```

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

```

3000 \skip_vertical:N \aboverulesep

```

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

```

3001 { \CT@arc@ \hrule height \heavyrulewidth }
3002 }
3003 { \@@_error_or_warning:n { bottomrule-without-booktabs } }
3004 }
3005 \l_@@_notes_code_after_tl
3006 \seq_gclear:N \g_@@_notes_seq
3007 \seq_gclear:N \g_@@_notes_in_caption_seq
3008 \int_gzero:N \c@tabularnote
3009 }

```

The following command will format (after the main tabular) one tabularnote (with the command `\item`). `#1` is the label (when the command `\tabularnote` has been used with an optional argument between square brackets) and `#2` is the text of the note. The second argument is provided by currying.

```

3010 \cs_set_protected:Npn \@@_one_tabularnote:nn #1
3011 {
3012   \tl_if_novalue:nTF { #1 }
3013   { \item }
3014   { \item [ \@@_notes_label_in_list:n { #1 } ] }
3015 }

```

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`.

```

3016 \cs_new_protected:Npn \@@_use_arraybox_with_notes_b:
3017 {
3018   \pgfpicture
3019     \@@_qpoint:n { row - 1 }
3020     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3021     \@@_qpoint:n { row - \int_use:N \c@iRow - base }
3022     \dim_gsub:Nn \g_tmpa_dim \pgf@y
3023   \endpgfpicture
3024   \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3025   \int_compare:nNnT \l_@@_first_row_int = 0
3026   {
3027     \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3028     \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3029   }
3030   \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3031 }

```

Now, the general case.

```

3032 \cs_new_protected:Npn \@@_use_arraybox_with_notes:
3033 {

```

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

```

3034   \tl_if_eq:NnT \l_@@_baseline_tl { t }
3035   { \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`.

```

3036   \pgfpicture
3037   \@@_qpoint:n { row - 1 }
3038   \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3039   \str_if_in:NnTF \l_@@_baseline_tl { line- }
3040   {
3041     \int_set:Nn \l_tmpa_int
3042     {
3043       \str_range:Nnn
3044         \l_@@_baseline_tl
3045         6
3046         { \tl_count:V \l_@@_baseline_tl }
3047     }
3048     \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
3049   }
3050   {
3051     \int_set:Nn \l_tmpa_int \l_@@_baseline_tl
3052     \bool_lazy_or:nnT
3053     { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
3054     { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
3055     {
3056       \@@_error:n { bad-value-for-baseline }
3057       \int_set:Nn \l_tmpa_int 1
3058     }
3059     \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }
3060   }
3061   \dim_gsub:Nn \g_tmpa_dim \pgf@y
3062   \endpgfpicture
3063   \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3064   \int_compare:nNnT \l_@@_first_row_int = 0
3065   {
3066     \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3067     \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3068   }
3069   \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }

```

```
3070 }
```

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.

```
3071 \cs_new_protected:Npn \@@_put_box_in_flow_bis:nn #1 #2
3072 {
```

We will compute the real width of both delimiters used.

```
3073   \dim_zero_new:N \l_@@_real_left_delim_dim
3074   \dim_zero_new:N \l_@@_real_right_delim_dim
3075   \hbox_set:Nn \l_tmpb_box
3076   {
3077     \c_math_toggle_token
3078     \left #1
3079     \vcenter
3080     {
3081       \vbox_to_ht:nn
3082       { \box_ht_plus_dp:N \l_tmpa_box }
3083       { }
3084     }
3085     \right .
3086     \c_math_toggle_token
3087   }
3088   \dim_set:Nn \l_@@_real_left_delim_dim
3089   { \box_wd:N \l_tmpb_box - \nulldelimiterspace }
3090   \hbox_set:Nn \l_tmpb_box
3091   {
3092     \c_math_toggle_token
3093     \left .
3094     \vbox_to_ht:nn
3095     { \box_ht_plus_dp:N \l_tmpa_box }
3096     { }
3097     \right #2
3098     \c_math_toggle_token
3099   }
3100   \dim_set:Nn \l_@@_real_right_delim_dim
3101   { \box_wd:N \l_tmpb_box - \nulldelimiterspace }
```

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

```
3102   \skip_horizontal:N \l_@@_left_delim_dim
3103   \skip_horizontal:N -\l_@@_real_left_delim_dim
3104   \@@_put_box_in_flow:
3105   \skip_horizontal:N \l_@@_right_delim_dim
3106   \skip_horizontal:N -\l_@@_real_right_delim_dim
3107 }
```

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).

```
3108 \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).

```
3109 {
3110   \peek_remove_spaces:n
3111   {
3112     \peek_meaning:NTF \end
3113     \@@_analyze_end:Nn
```

```

3114         {
3115             \@@_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).
3116             \@@_array:V \g_@@_preamble_tl
3117         }
3118     }
3119 }
3120 {
3121     \@@_create_col_nodes:
3122     \endarray
3123 }

```

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

```

3124 \NewDocumentEnvironment { @@-light-syntax } { b }
3125 {

```

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`.

```

3126     \tl_if_empty:nT { #1 } { \@@_fatal:n { empty-environment } }
3127     \tl_map_inline:nn { #1 }
3128     {
3129         \str_if_eq:nnT { ##1 } { & }
3130         { \@@_fatal:n { ampersand-in-light-syntax } }
3131         \str_if_eq:nnT { ##1 } { \ }
3132         { \@@_fatal:n { double-backslash-in-light-syntax } }
3133     }

```

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`.

```

3134     \@@_light_syntax_i:w #1 \CodeAfter \q_stop

```

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

```

3135 }

```

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.

```

3136 {
3137     \@@_create_col_nodes:
3138     \endarray
3139 }

3140 \cs_new_protected:Npn \@@_light_syntax_i:w #1\CodeAfter #2\q_stop
3141 {
3142     \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).

```

3143     \seq_clear_new:N \l_@@_rows_seq

```

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

```

3144     \tl_set_rescan:Nno \l_@@_end_of_row_tl { } \l_@@_end_of_row_tl
3145     \seq_set_split:NVn \l_@@_rows_seq \l_@@_end_of_row_tl { #1 }

```

We delete the last row if it is empty.

```

3146     \seq_pop_right:NN \l_@@_rows_seq \l_tmpa_tl
3147     \tl_if_empty:NF \l_tmpa_tl
3148     { \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.

```

3149   \int_compare:nNnT \l_@@_last_row_int = { -1 }
3150   { \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`).

```

3151   \tl_clear_new:N \l_@@_new_body_tl
3152   \int_zero_new:N \l_@@_nb_cols_int

```

First, we treat the first row.

```

3153   \seq_pop_left:NN \l_@@_rows_seq \l_tmpa_tl
3154   \@@_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).

```

3155   \seq_map_inline:Nn \l_@@_rows_seq
3156   {
3157     \tl_put_right:Nn \l_@@_new_body_tl { \\ }
3158     \@@_line_with_light_syntax:n { ##1 }
3159   }

3160   \int_compare:nNnT \l_@@_last_col_int = { -1 }
3161   {
3162     \int_set:Nn \l_@@_last_col_int
3163     { \l_@@_nb_cols_int - 1 + \l_@@_first_col_int }
3164   }

```

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.

```

3165   \@@_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`).

```

3166   \@@_array:V \g_@@_preamble_tl \l_@@_new_body_tl
3167   }

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

```

3184   \cs_new_protected:Npn \@@_analyze_end:Nn #1 #2
3185   {
3186     \str_if_eq:VnT \g_@@_name_env_str { #2 }
3187     { \@@_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.

```
3188     \end { #2 }
3189 }
```

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).

```
3190 \cs_new:Npn \@@_create_col_nodes:
3191 {
3192   \crrc
3193   \int_compare:nNnT \l_@@_first_col_int = 0
3194   {
3195     \omit
3196     \hbox_overlap_left:n
3197     {
3198       \bool_if:NT \l_@@_code_before_bool
3199       { \pgfsys@markposition { \@@_env: - col - 0 } }
3200       \pgfpicture
3201       \pgfrememberpicturepositiononpagetrue
3202       \pgfcoordinate { \@@_env: - col - 0 } \pgfpointorigin
3203       \str_if_empty:NF \l_@@_name_str
3204       { \pgfnodealias { \l_@@_name_str - col - 0 } { \@@_env: - col - 0 } }
3205       \endpgfpicture
3206       \skip_horizontal:N 2\col@sep
3207       \skip_horizontal:N \g_@@_width_first_col_dim
3208     }
3209     &
3210   }
3211   \omit
```

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

```
3212   \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`).

```
3213   \int_compare:nNnTF \l_@@_first_col_int = 0
3214   {
3215     \bool_if:NT \l_@@_code_before_bool
3216     {
3217       \hbox
3218       {
3219         \skip_horizontal:N -0.5\arrayrulewidth
3220         \pgfsys@markposition { \@@_env: - col - 1 }
3221         \skip_horizontal:N 0.5\arrayrulewidth
3222       }
3223     }
3224     \pgfpicture
3225     \pgfrememberpicturepositiononpagetrue
3226     \pgfcoordinate { \@@_env: - col - 1 }
3227     { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3228     \str_if_empty:NF \l_@@_name_str
3229     { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
3230     \endpgfpicture
3231   }
3232   {
3233     \bool_if:NT \l_@@_code_before_bool
3234     {
3235       \hbox
3236       {
3237         \skip_horizontal:N 0.5\arrayrulewidth
3238         \pgfsys@markposition { \@@_env: - col - 1 }
3239         \skip_horizontal:N -0.5\arrayrulewidth
```

```

3240     }
3241   }
3242   \pgfpicture
3243   \pgfrememberpicturepositiononpagetrue
3244   \pgfcoordinate { \@@_env: - col - 1 }
3245   { \pgfpoint { 0.5 \arrayrulewidth } \c_zero_dim }
3246   \str_if_empty:NF \l_@@_name_str
3247   { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
3248   \endpgfpicture
3249 }

```

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.

```

3250   \skip_gset:Nn \g_tmpa_skip { 0 pt+plus 1 fill }
3251   \bool_if:NF \l_@@_auto_columns_width_bool
3252   { \dim_compare:nNnT \l_@@_columns_width_dim > \c_zero_dim }
3253   {
3254     \bool_lazy_and:nnTF
3255     \l_@@_auto_columns_width_bool
3256     { \bool_not_p:n \l_@@_block_auto_columns_width_bool }
3257     { \skip_gset_eq:NN \g_tmpa_skip \g_@@_max_cell_width_dim }
3258     { \skip_gset_eq:NN \g_tmpa_skip \l_@@_columns_width_dim }
3259     \skip_gadd:Nn \g_tmpa_skip { 2 \col@sep }
3260   }
3261   \skip_horizontal:N \g_tmpa_skip
3262   \hbox
3263   {
3264     \bool_if:NT \l_@@_code_before_bool
3265     {
3266       \hbox
3267       {
3268         \skip_horizontal:N -0.5\arrayrulewidth
3269         \pgfsys@markposition { \@@_env: - col - 2 }
3270         \skip_horizontal:N 0.5\arrayrulewidth
3271       }
3272     }
3273     \pgfpicture
3274     \pgfrememberpicturepositiononpagetrue
3275     \pgfcoordinate { \@@_env: - col - 2 }
3276     { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3277     \str_if_empty:NF \l_@@_name_str
3278     { \pgfnodealias { \l_@@_name_str - col - 2 } { \@@_env: - col - 2 } }
3279     \endpgfpicture
3280   }

```

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.

```

3281   \int_gset:Nn \g_tmpa_int 1
3282   \bool_if:NTF \g_@@_last_col_found_bool
3283   { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 3 } 0 } }
3284   { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 2 } 0 } }
3285   {
3286     &
3287     \omit
3288     \int_gincr:N \g_tmpa_int

```

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

```

3289     \skip_horizontal:N \g_tmpa_skip
3290     \bool_if:NT \l_@@_code_before_bool
3291     {

```

```

3292         \hbox
3293         {
3294             \skip_horizontal:N -0.5\arrayrulewidth
3295             \pgfsys@markposition
3296             { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3297             \skip_horizontal:N 0.5\arrayrulewidth
3298         }
3299     }

```

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

```

3300     \pgfpicture
3301     \pgfrememberpicturepositiononpagetrue
3302     \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3303     { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3304     \str_if_empty:NF \l_@@_name_str
3305     {
3306         \pgfnodealias
3307         { \l_@@_name_str - col - \int_eval:n { \g_tmpa_int + 1 } }
3308         { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3309     }
3310     \endpgfpicture
3311 }

```

```

3312     &
3313     \omit

```

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

```

3314     \int_compare:nNnT \g_@@_col_total_int = 1
3315     { \skip_gset:Nn \g_tmpa_skip { 0 pt~plus 1 fill } }
3316     \skip_horizontal:N \g_tmpa_skip
3317     \int_gincr:N \g_tmpa_int
3318     \bool_lazy_all:nT
3319     {
3320         { \bool_not_p:n \g_@@_delims_bool }
3321         { \bool_not_p:n \l_@@_tabular_bool }
3322         { \clist_if_empty_p:N \l_@@_vlines_clist }
3323         { \bool_not_p:n \l_@@_exterior_arraycolsep_bool }
3324         { ! \l_@@_bar_at_end_of_pream_bool }
3325     }
3326     { \skip_horizontal:N -\col@sep }
3327     \bool_if:NT \l_@@_code_before_bool
3328     {
3329         \hbox
3330         {
3331             \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.

```

3332         \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3333         { \skip_horizontal:N -\arraycolsep }
3334         \pgfsys@markposition
3335         { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3336         \skip_horizontal:N 0.5\arrayrulewidth
3337         \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3338         { \skip_horizontal:N \arraycolsep }
3339     }
3340 }
3341 \pgfpicture
3342 \pgfrememberpicturepositiononpagetrue
3343 \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3344 {
3345     \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool

```

```

3346         {
3347             \pgfpoint
3348             { - 0.5 \arrayrulewidth - \arraycolsep }
3349             \c_zero_dim
3350         }
3351         { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3352     }
3353     \str_if_empty:NF \l_@@_name_str
3354     {
3355         \pgfnodealias
3356         { \l_@@_name_str - col - \int_eval:n { \g_tmpa_int + 1 } }
3357         { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3358     }
3359     \endpgfpicture

3360     \bool_if:NT \g_@@_last_col_found_bool
3361     {
3362         \hbox_overlap_right:n
3363         {
3364             \skip_horizontal:N \g_@@_width_last_col_dim
3365             \bool_if:NT \l_@@_code_before_bool
3366             {
3367                 \pgfsys@markposition
3368                 { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3369             }
3370             \pgfpicture
3371             \pgfrememberpicturerepositiononpagetrue
3372             \pgfcoordinate
3373             { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3374             \pgfpointorigin
3375             \str_if_empty:NF \l_@@_name_str
3376             {
3377                 \pgfnodealias
3378                 {
3379                     \l_@@_name_str - col
3380                     - \int_eval:n { \g_@@_col_total_int + 1 }
3381                 }
3382                 { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3383             }
3384             \endpgfpicture
3385         }
3386     }
3387     \cr
3388 }

```

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

```

3389 \tl_const:Nn \c_@@_preamble_first_col_tl
3390 {
3391     >
3392     {

```

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).

```

3393     \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
3394     \bool_gset_true:N \g_@@_after_col_zero_bool
3395     \@@_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.

```

3396     \hbox_set:Nw \l_@@_cell_box
3397     \@@_math_toggle_token:
3398     \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”.

```

3399     \bool_lazy_and:nnT
3400     { \int_compare_p:nNn \c@iRow > 0 }
3401     {
3402         \bool_lazy_or_p:nn
3403         { \int_compare_p:nNn \l_@@_last_row_int < 0 }
3404         { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
3405     }
3406     {
3407         \l_@@_code_for_first_col_tl
3408         \xglobal \colorlet { nicematrix-first-col } { . }
3409     }
3410 }

```

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`.

```

3411     l
3412     <
3413     {
3414         \@@_math_toggle_token:
3415         \hbox_set_end:
3416         \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
3417         \@@_adjust_size_box:
3418         \@@_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.

```

3419     \dim_gset:Nn \g_@@_width_first_col_dim
3420     { \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.

```

3421     \hbox_overlap_left:n
3422     {
3423         \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > \c_zero_dim
3424         \@@_node_for_cell:
3425         { \box_use_drop:N \l_@@_cell_box }
3426         \skip_horizontal:N \l_@@_left_delim_dim
3427         \skip_horizontal:N \l_@@_left_margin_dim
3428         \skip_horizontal:N \l_@@_extra_left_margin_dim
3429     }
3430     \bool_gset_false:N \g_@@_empty_cell_bool
3431     \skip_horizontal:N -2\col@sep
3432 }
3433 }

```

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

```

3434 \tl_const:Nn \c_@@_preamble_last_col_tl
3435 {
3436     >
3437     {
3438         \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).

```

3439     \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.

```

3440     \bool_gset_true:N \g_@@_last_col_found_bool
3441     \int_gincr:N \c@jCol
3442     \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.

```

3443     \hbox_set:Nw \l_@@_cell_box
3444     \@@_math_toggle_token:

```

```
3445 \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”.

```
3446 \int_compare:nNnT \c@iRow > 0
3447 {
3448   \bool_lazy_or:nnT
3449   { \int_compare_p:nNn \l_@@_last_row_int < 0 }
3450   { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
3451   {
3452     \l_@@_code_for_last_col_tl
3453     \xglobal \colorlet { nicematrix-last-col } { . }
3454   }
3455 }
3456 }
3457 1
3458 <
3459 {
3460   \@@_math_toggle_token:
3461   \hbox_set_end:
3462   \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
3463   \@@_adjust_size_box:
3464   \@@_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.

```
3465 \dim_gset:Nn \g_@@_width_last_col_dim
3466 { \dim_max:nn \g_@@_width_last_col_dim { \box_wd:N \l_@@_cell_box } }
3467 \skip_horizontal:N -2\col@sep
```

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

```
3468 \hbox_overlap_right:n
3469 {
3470   \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \c_zero_dim
3471   {
3472     \skip_horizontal:N \l_@@_right_delim_dim
3473     \skip_horizontal:N \l_@@_right_margin_dim
3474     \skip_horizontal:N \l_@@_extra_right_margin_dim
3475     \@@_node_for_cell:
3476   }
3477 }
3478 \bool_gset_false:N \g_@@_empty_cell_bool
3479 }
3480 }
```

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

```
3481 \NewDocumentEnvironment { NiceArray } { }
3482 {
3483   \bool_gset_false:N \g_@@_delims_bool
3484   \str_if_empty:NT \g_@@_name_env_str
3485   { \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).

```
3486 \NiceArrayWithDelims . .
3487 }
3488 { \endNiceArrayWithDelims }
```

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

```
3489 \cs_new_protected:Npn \@@_def_env:nnn #1 #2 #3
3490 {
3491   \NewDocumentEnvironment { #1 NiceArray } { }
```

```

3492 {
3493   \bool_gset_true:N \g_@@_delims_bool
3494   \str_if_empty:NT \g_@@_name_env_str
3495   { \str_gset:Nn \g_@@_name_env_str { #1 NiceArray } }
3496   \@@_test_if_math_mode:
3497   \NiceArrayWithDelims #2 #3
3498 }
3499 { \endNiceArrayWithDelims }
3500 }
3501 \@@_def_env:nnn p ( )
3502 \@@_def_env:nnn b [ ]
3503 \@@_def_env:nnn B \{ \}
3504 \@@_def_env:nnn v | |
3505 \@@_def_env:nnn V \| \|

```

14 The environment `{NiceMatrix}` and its variants

```

3506 \cs_new_protected:Npn \@@_begin_of_NiceMatrix:nn #1 #2
3507 {
3508   \bool_set_false:N \l_@@_preamble_bool
3509   \tl_clear:N \l_tmpa_tl
3510   \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3511   { \tl_set:Nn \l_tmpa_tl { @ { } } }
3512   \tl_put_right:Nn \l_tmpa_tl
3513   {
3514     *
3515     {
3516       \int_case:nnF \l_@@_last_col_int
3517       {
3518         { -2 } { \c@MaxMatrixCols }
3519         { -1 } { \int_eval:n { \c@MaxMatrixCols + 1 } }

```

The value 0 can't occur here since we are in a matrix (which is an environment without preamble).

```

3520   }
3521   { \int_eval:n { \l_@@_last_col_int - 1 } }
3522 }
3523 { #2 }
3524 }
3525 \tl_set:Nn \l_tmpb_tl { \use:c { #1 NiceArray } }
3526 \exp_args:NV \l_tmpb_tl \l_tmpa_tl
3527 }
3528 \cs_generate_variant:Nn \@@_begin_of_NiceMatrix:nn { n V }
3529 \clist_map_inline:nn { p , b , B , v , V }
3530 {
3531   \NewDocumentEnvironment { #1 NiceMatrix } { ! O { } }
3532   {
3533     \bool_gset_true:N \g_@@_delims_bool
3534     \str_gset:Nn \g_@@_name_env_str { #1 NiceMatrix }
3535     \keys_set:nn { NiceMatrix / NiceMatrix } { ##1 }
3536     \@@_begin_of_NiceMatrix:nV { #1 } \l_@@_columns_type_tl
3537   }
3538   { \use:c { end #1 NiceArray } }
3539 }

```

We define also an environment `{NiceMatrix}`

```

3540 \NewDocumentEnvironment { NiceMatrix } { ! O { } }
3541 {
3542   \str_gset:Nn \g_@@_name_env_str { NiceMatrix }

```

```

3543 \keys_set:nn { NiceMatrix / NiceMatrix } { #1 }
3544 \bool_lazy_or:nnT
3545   { \clist_if_empty_p:N \l_@@_vlines_clist }
3546   { \l_@@_except_borders_bool }
3547   { \bool_set_true:N \l_@@_NiceMatrix_without_vlines_bool }
3548 \@@_begin_of_NiceMatrix:nV { } \l_@@_columns_type_tl
3549 }
3550 { \endNiceArray }

```

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

```

3551 \cs_new_protected:Npn \@@_NotEmpty:
3552   { \bool_gset_true:N \g_@@_not_empty_cell_bool }

```

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

```

3553 \NewDocumentEnvironment { NiceTabular } { 0 { } m ! 0 { } }
3554 {

```

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`.

```

3555   \dim_compare:nNnT \l_@@_width_dim = \c_zero_dim
3556     { \dim_set_eq:NN \l_@@_width_dim \linewidth }
3557   \str_gset:Nn \g_@@_name_env_str { NiceTabular }
3558   \keys_set:nn { NiceMatrix / NiceTabular } { #1 , #3 }
3559   \tl_if_empty:NF \l_@@_short_caption_tl
3560     {
3561       \tl_if_empty:NT \l_@@_caption_tl
3562         {
3563           \@@_error_or_warning:n { short-caption-without-caption }
3564           \tl_set_eq:NN \l_@@_caption_tl \l_@@_short_caption_tl
3565         }
3566     }
3567   \tl_if_empty:NF \l_@@_label_tl
3568     {
3569       \tl_if_empty:NT \l_@@_caption_tl
3570       { \@@_error_or_warning:n { label-without-caption } }
3571     }
3572   \NewDocumentEnvironment { TabularNote } { b }
3573   {
3574     \bool_if:NTF \l_@@_in_code_after_bool
3575       { \@@_error_or_warning:n { TabularNote-in-CodeAfter } }
3576       {
3577         \tl_if_empty:NF \g_@@_tabularnote_tl
3578         { \tl_gput_right:Nn \g_@@_tabularnote_tl { \par } }
3579         \tl_gput_right:Nn \g_@@_tabularnote_tl { ##1 }
3580       }
3581     }
3582   { }
3583   \bool_set_true:N \l_@@_tabular_bool
3584   \NiceArray { #2 }
3585 }
3586 { \endNiceArray }

3587 \cs_set_protected:Npn \@@_newcolumnntype #1
3588 {
3589   \cs_if_free:cT { NC @ find @ #1 }
3590   { \NC@list \expandafter { \the \NC@list \NC@do #1 } }
3591   \cs_set:cpn { NC @ find @ #1 } ##1 #1 { \NC@ { ##1 } }
3592   \peek_meaning:NTF [
3593     { \newcol@ #1 }
3594     { \newcol@ #1 [ 0 ] }
3595   }

```

```

3596 \NewDocumentEnvironment { NiceTabularX } { m O { } m ! O { } }
3597 {

```

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}`).

```

3598 \IfPackageLoadedTF { tabularx }
3599 { \newcolumnntype { X } { \@@_X } }
3600 { }
3601 \str_gset:Nn \g_@@_name_env_str { NiceTabularX }
3602 \dim_zero_new:N \l_@@_width_dim
3603 \dim_set:Nn \l_@@_width_dim { #1 }
3604 \keys_set:nn { NiceMatrix / NiceTabular } { #2 , #4 }
3605 \bool_set_true:N \l_@@_tabular_bool
3606 \NiceArray { #3 }
3607 }
3608 { \endNiceArray }

3609 \NewDocumentEnvironment { NiceTabular* } { m O { } m ! O { } }
3610 {
3611 \str_gset:Nn \g_@@_name_env_str { NiceTabular* }
3612 \dim_set:Nn \l_@@_tabular_width_dim { #1 }
3613 \keys_set:nn { NiceMatrix / NiceTabular } { #2 , #4 }
3614 \bool_set_true:N \l_@@_tabular_bool
3615 \NiceArray { #3 }
3616 }
3617 { \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`).

```

3618 \cs_new_protected:Npn \@@_deal_with_rounded_corners:
3619 {
3620 \bool_lazy_all:nT
3621 {
3622 { \int_compare_p:nNn \l_@@_tab_rounded_corners_dim > \c_zero_dim }
3623 \l_@@_hvlines_bool
3624 { ! \g_@@_delims_bool }
3625 { ! \l_@@_except_borders_bool }
3626 }
3627 {
3628 \bool_set_true:N \l_@@_except_borders_bool
3629 \clist_if_empty:NF \l_@@_corners_clist
3630 { \@@_error:n { hvlines,~rounded-corners~and~corners } }
3631 \tl_gput_right:Nn \g_@@_pre_code_after_tl
3632 {
3633 \@@_stroke_block:nnn
3634 {
3635 rounded-corners = \dim_use:N \l_@@_tab_rounded_corners_dim ,
3636 draw = \l_@@_rules_color_tl
3637 }
3638 { 1-1 }
3639 { \int_use:N \c@iRow - \int_use:N \c@jCol }
3640 }
3641 }
3642 }

3643 \cs_new_protected:Npn \@@_after_array:
3644 {
3645 \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.

```
3646 \bool_if:NT \g_@@_last_col_found_bool
3647 { \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`.

```
3648 \bool_if:NT \l_@@_last_col_without_value_bool
3649 { \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.

```
3650 \bool_if:NT \l_@@_last_row_without_value_bool
3651 { \int_set_eq:NN \l_@@_last_row_int \g_@@_row_total_int }
```

```
3652 \tl_gput_right:Nx \g_@@_aux_tl
3653 {
3654   \seq_gset_from_clist:Nn \exp_not:N \g_@@_size_seq
3655   {
3656     \int_use:N \l_@@_first_row_int ,
3657     \int_use:N \c@iRow ,
3658     \int_use:N \g_@@_row_total_int ,
3659     \int_use:N \l_@@_first_col_int ,
3660     \int_use:N \c@jCol ,
3661     \int_use:N \g_@@_col_total_int
3662   }
3663 }
```

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`).

```
3664 \seq_if_empty:NF \g_@@_pos_of_blocks_seq
3665 {
3666   \tl_gput_right:Nx \g_@@_aux_tl
3667   {
3668     \seq_gset_from_clist:Nn \exp_not:N \g_@@_pos_of_blocks_seq
3669     { \seq_use:Nnnn \g_@@_pos_of_blocks_seq , , , }
3670   }
3671 }
3672 \seq_if_empty:NF \g_@@_multicolumn_cells_seq
3673 {
3674   \tl_gput_right:Nx \g_@@_aux_tl
3675   {
3676     \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_cells_seq
3677     { \seq_use:Nnnn \g_@@_multicolumn_cells_seq , , , }
3678     \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_sizes_seq
3679     { \seq_use:Nnnn \g_@@_multicolumn_sizes_seq , , , }
3680   }
3681 }
```

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

```
3682 \@@_create_diag_nodes:
```

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

```
3683 \pgfpicture
3684 \int_step_inline:nn \c@iRow
3685 {
3686   \pgfnodealias
3687   { \@@_env: - ##1 - last }
3688   { \@@_env: - ##1 - \int_use:N \c@jCol }
```

```

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

```

3712     \bool_if:NT \l_@@_parallelize_diags_bool
3713     {
3714         \int_gzero_new:N \g_@@_ddots_int
3715         \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.

```

3716         \dim_gzero_new:N \g_@@_delta_x_one_dim
3717         \dim_gzero_new:N \g_@@_delta_y_one_dim
3718         \dim_gzero_new:N \g_@@_delta_x_two_dim
3719         \dim_gzero_new:N \g_@@_delta_y_two_dim
3720     }
3721     \int_zero_new:N \l_@@_initial_i_int
3722     \int_zero_new:N \l_@@_initial_j_int
3723     \int_zero_new:N \l_@@_final_i_int
3724     \int_zero_new:N \l_@@_final_j_int
3725     \bool_set_false:N \l_@@_initial_open_bool
3726     \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.

```

3727     \bool_if:NT \l_@@_small_bool
3728     {
3729         \dim_set:Nn \l_@@_xdots_radius_dim { 0.7 \l_@@_xdots_radius_dim }
3730         \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.

```

3731         \dim_set:Nn \l_@@_xdots_shorten_start_dim
3732         { 0.6 \l_@@_xdots_shorten_start_dim }
3733         \dim_set:Nn \l_@@_xdots_shorten_end_dim

```

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

```

3734         { 0.6 \l_@@_xdots_shorten_end_dim }
3735     }

```

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

```

3736 \@@_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.

```

3737 \@@_compute_corners:

```

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

```

3738 \@@_adjust_pos_of_blocks_seq:
3739 \@@_deal_with_rounded_corners:
3740 \tl_if_empty:NF \l_@@_hlines_clist \@@_draw_hlines:
3741 \tl_if_empty:NF \l_@@_vlines_clist \@@_draw_vlines:

```

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

```

3742 \IfPackageLoadedTF { tikz }
3743 {
3744     \tikzset
3745     {
3746         every~picture / .style =
3747         {
3748             overlay ,
3749             remember~picture ,
3750             name~prefix = \@@_env: -
3751         }
3752     }
3753 }
3754 { }
3755 \cs_set_eq:NN \ialign \@@_old_ialign:
3756 \cs_set_eq:NN \SubMatrix \@@_SubMatrix
3757 \cs_set_eq:NN \UnderBrace \@@_UnderBrace
3758 \cs_set_eq:NN \OverBrace \@@_OverBrace
3759 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
3760 \cs_set_eq:NN \line \@@_line
3761 \g_@@_pre_code_after_tl
3762 \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.

```

3763 \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).

```

3764 \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`).

```

3765 \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
3766 { \@@_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:`.

```

3767 \bool_set_true:N \l_@@_in_code_after_bool
3768 \exp_last_unbraced:N \@@_CodeAfter_keys: \g_nicematrix_code_after_tl
3769 \scan_stop:
3770 \tl_gclear:N \g_nicematrix_code_after_tl
3771 \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.

```

3772 \tl_if_empty:NF \g_@@_pre_code_before_tl
3773 {
3774   \tl_gput_right:Nx \g_@@_aux_tl
3775   {
3776     \tl_gset:Nn \exp_not:N \g_@@_pre_code_before_tl
3777     { \exp_not:V \g_@@_pre_code_before_tl }
3778   }
3779   \tl_gclear:N \g_@@_pre_code_before_tl
3780 }
3781 \tl_if_empty:NF \g_nicematrix_code_before_tl
3782 {
3783   \tl_gput_right:Nx \g_@@_aux_tl
3784   {
3785     \tl_gset:Nn \exp_not:N \g_@@_code_before_tl
3786     { \exp_not:V \g_nicematrix_code_before_tl }
3787   }
3788   \tl_gclear:N \g_nicematrix_code_before_tl
3789 }
3790 \str_gclear:N \g_@@_name_env_str
3791 \@@_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`.

```

3792 \cs_gset_eq:NN \CT@arc@ \@@_old_CT@arc@
3793 }

```

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`.

```

3794 \NewDocumentCommand \@@_CodeAfter_keys: { 0 { } }
3795 { \keys_set:nn { NiceMatrix / CodeAfter } { #1 } }

```

We remind that the first mandatory argument of the command `\Block` is the size of the block with the special format *i-j*. However, the user is allowed to omit *i* or *j* (or both). This will be interpreted as: the last row (resp. column) of the block will be the last row (resp. column) of the block (without the potential exterior row—resp. column—of the array). By convention, this is stored in `\g_@@_pos_of_blocks_seq` (and `\g_@@_blocks_seq`) as a number of rows (resp. columns) for the block equal to 100. It's possible, after the construction of the array, to replace these values by the correct ones (since we know the number of rows and columns of the array).

```

3796 \cs_new_protected:Npn \@@_adjust_pos_of_blocks_seq:
3797 {
3798   \seq_gset_map_x:NNn \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq
3799   { \@@_adjust_pos_of_blocks_seq_i:nnnnn ##1 }
3800 }

```

¹²e.g. `\color[rgb]{0.5,0.5,0}`

The following command must *not* be protected.

```

3801 \cs_new:Npn \@@_adjust_pos_of_blocks_seq_i:nnnnn #1 #2 #3 #4 #5
3802 {
3803   { #1 }
3804   { #2 }
3805   {
3806     \int_compare:nNnTF { #3 } > { 99 }
3807       { \int_use:N \c@iRow }
3808       { #3 }
3809   }
3810   {
3811     \int_compare:nNnTF { #4 } > { 99 }
3812       { \int_use:N \c@jCol }
3813       { #4 }
3814   }
3815   { #5 }
3816 }

```

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).

```

3817 \hook_gput_code:nnn { begindocument } { . }
3818 {
3819   \cs_new_protected:Npx \@@_draw_dotted_lines:
3820   {
3821     \c_@@_pgfortikzpicture_tl
3822     \@@_draw_dotted_lines_i:
3823     \c_@@_endpgfortikzpicture_tl
3824   }
3825 }

```

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

```

3826 \cs_new_protected:Npn \@@_draw_dotted_lines_i:
3827 {
3828   \pgfrememberpicturepositiononpagetrue
3829   \pgf@relevantforpicturesizefalse
3830   \g_@@_HVdotsfor_lines_tl
3831   \g_@@_Vdots_lines_tl
3832   \g_@@_Ddots_lines_tl
3833   \g_@@_Iddots_lines_tl
3834   \g_@@_Cdots_lines_tl
3835   \g_@@_Ldots_lines_tl
3836 }

3837 \cs_new_protected:Npn \@@_restore_iRow_jCol:
3838 {
3839   \cs_if_exist:NT \theiRow { \int_gset_eq:NN \c@iRow \l_@@_old_iRow_int }
3840   \cs_if_exist:NT \thejCol { \int_gset_eq:NN \c@jCol \l_@@_old_jCol_int }
3841 }

```

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

```

3842 \pgfdeclareshape { @@_diag_node }
3843 {
3844   \savedanchor { \five }
3845   {
3846     \dim_gset_eq:NN \pgf@x \l_tmpa_dim
3847     \dim_gset_eq:NN \pgf@y \l_tmpb_dim
3848   }
3849   \anchor { 5 } { \five }

```

```

3850   \anchor { center } { \pgfpointorigin }
3851 }

```

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).

```

3852 \cs_new_protected:Npn \@@_create_diag_nodes:
3853 {
3854   \pgfpicture
3855   \pgfrememberpicturepositiononpagetrue
3856   \int_step_inline:nn { \int_max:nn \c@iRow \c@jCol }
3857   {
3858     \@@_qpoint:n { col - \int_min:nn { ##1 } { \c@jCol + 1 } }
3859     \dim_set_eq:NN \l_tmpa_dim \pgf@x
3860     \@@_qpoint:n { row - \int_min:nn { ##1 } { \c@iRow + 1 } }
3861     \dim_set_eq:NN \l_tmpb_dim \pgf@y
3862     \@@_qpoint:n { col - \int_min:nn { ##1 + 1 } { \c@jCol + 1 } }
3863     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
3864     \@@_qpoint:n { row - \int_min:nn { ##1 + 1 } { \c@iRow + 1 } }
3865     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
3866     \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.

```

3867     \dim_set:Nn \l_tmpa_dim { ( \l_@@_tmpc_dim - \l_tmpa_dim ) / 2 }
3868     \dim_set:Nn \l_tmpb_dim { ( \l_@@_tmpd_dim - \l_tmpb_dim ) / 2 }
3869     \pgfnode { @_diag_node } { center } { } { \@@_env: - ##1 } { }
3870     \str_if_empty:NF \l_@@_name_str
3871     { \pgfnodealias { \l_@@_name_str - ##1 } { \@@_env: - ##1 } }
3872   }

```

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

```

3873   \int_set:Nn \l_tmpa_int { \int_max:nn \c@iRow \c@jCol + 1 }
3874   \@@_qpoint:n { row - \int_min:nn { \l_tmpa_int } { \c@iRow + 1 } }
3875   \dim_set_eq:NN \l_tmpa_dim \pgf@y
3876   \@@_qpoint:n { col - \int_min:nn { \l_tmpa_int } { \c@jCol + 1 } }
3877   \pgfcoordinate
3878   { \@@_env: - \int_use:N \l_tmpa_int } { \pgfpoint \pgf@x \l_tmpa_dim }
3879   \pgfnodealias
3880   { \@@_env: - last }
3881   { \@@_env: - \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
3882   \str_if_empty:NF \l_@@_name_str
3883   {
3884     \pgfnodealias
3885     { \l_@@_name_str - \int_use:N \l_tmpa_int }
3886     { \@@_env: - \int_use:N \l_tmpa_int }
3887     \pgfnodealias
3888     { \l_@@_name_str - last }
3889     { \@@_env: - last }
3890   }
3891   \endpgfpicture
3892 }

```

17 We draw the dotted lines

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

its right.

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

The command `\l_@@_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.

```
3893 \cs_new_protected:Npn \l_@@_find_extremities_of_line:nnnn #1 #2 #3 #4
3894 {
```

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

```
3895 \cs_set:cpn { @@ _ dotted _ #1 - #2 } { }
```

Initialization of variables.

```
3896 \int_set:Nn \l_@@_initial_i_int { #1 }
3897 \int_set:Nn \l_@@_initial_j_int { #2 }
3898 \int_set:Nn \l_@@_final_i_int { #1 }
3899 \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.

```
3900 \bool_set_false:N \l_@@_stop_loop_bool
3901 \bool_do_until:Nn \l_@@_stop_loop_bool
3902 {
3903   \int_add:Nn \l_@@_final_i_int { #3 }
3904   \int_add:Nn \l_@@_final_j_int { #4 }
```

We test if we are still in the matrix.

```
3905   \bool_set_false:N \l_@@_final_open_bool
3906   \int_compare:nNnTF \l_@@_final_i_int > \l_@@_row_max_int
3907   {
3908     \int_compare:nNnTF { #3 } = 1
3909     { \bool_set_true:N \l_@@_final_open_bool }
3910     {
3911       \int_compare:nNnTF \l_@@_final_j_int > \l_@@_col_max_int
3912       { \bool_set_true:N \l_@@_final_open_bool }
3913     }
3914   }
3915   {
3916     \int_compare:nNnTF \l_@@_final_j_int < \l_@@_col_min_int
3917     {
3918       \int_compare:nNnTF { #4 } = { -1 }
3919       { \bool_set_true:N \l_@@_final_open_bool }
3920     }
3921   }
```

```

3922         \int_compare:nNtT \l_@@_final_j_int > \l_@@_col_max_int
3923         {
3924             \int_compare:nNtT { #4 } = 1
3925             { \bool_set_true:N \l_@@_final_open_bool }
3926         }
3927     }
3928 }
3929 \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.

```

3930 {

```

We do a step backwards.

```

3931     \int_sub:Nn \l_@@_final_i_int { #3 }
3932     \int_sub:Nn \l_@@_final_j_int { #4 }
3933     \bool_set_true:N \l_@@_stop_loop_bool
3934 }

```

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`.

```

3935 {
3936     \cs_if_exist:cTF
3937     {
3938         @@ _ dotted _
3939         \int_use:N \l_@@_final_i_int -
3940         \int_use:N \l_@@_final_j_int
3941     }
3942     {
3943         \int_sub:Nn \l_@@_final_i_int { #3 }
3944         \int_sub:Nn \l_@@_final_j_int { #4 }
3945         \bool_set_true:N \l_@@_final_open_bool
3946         \bool_set_true:N \l_@@_stop_loop_bool
3947     }
3948     {
3949         \cs_if_exist:cTF
3950         {
3951             pgf @ sh @ ns @ \@@_env:
3952             - \int_use:N \l_@@_final_i_int
3953             - \int_use:N \l_@@_final_j_int
3954         }
3955         { \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.

```

3956     {
3957         \cs_set:cpn
3958         {
3959             @@ _ dotted _
3960             \int_use:N \l_@@_final_i_int -
3961             \int_use:N \l_@@_final_j_int
3962         }
3963         { }
3964     }
3965 }
3966 }
3967 }

```

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

```

3968     \bool_set_false:N \l_@@_stop_loop_bool

```

```

3969 \bool_do_until:Nn \l_@@_stop_loop_bool
3970 {
3971   \int_sub:Nn \l_@@_initial_i_int { #3 }
3972   \int_sub:Nn \l_@@_initial_j_int { #4 }
3973   \bool_set_false:N \l_@@_initial_open_bool
3974   \int_compare:nNnTF \l_@@_initial_i_int < \l_@@_row_min_int
3975   {
3976     \int_compare:nNnTF { #3 } = 1
3977     { \bool_set_true:N \l_@@_initial_open_bool }
3978     {
3979       \int_compare:nNnT \l_@@_initial_j_int = { \l_@@_col_min_int -1 }
3980       { \bool_set_true:N \l_@@_initial_open_bool }
3981     }
3982   }
3983   {
3984     \int_compare:nNnTF \l_@@_initial_j_int < \l_@@_col_min_int
3985     {
3986       \int_compare:nNnT { #4 } = 1
3987       { \bool_set_true:N \l_@@_initial_open_bool }
3988     }
3989     {
3990       \int_compare:nNnT \l_@@_initial_j_int > \l_@@_col_max_int
3991       {
3992         \int_compare:nNnT { #4 } = { -1 }
3993         { \bool_set_true:N \l_@@_initial_open_bool }
3994       }
3995     }
3996   }
3997   \bool_if:NnTF \l_@@_initial_open_bool
3998   {
3999     \int_add:Nn \l_@@_initial_i_int { #3 }
4000     \int_add:Nn \l_@@_initial_j_int { #4 }
4001     \bool_set_true:N \l_@@_stop_loop_bool
4002   }
4003   {
4004     \cs_if_exist:cTF
4005     {
4006       @@ _ dotted _
4007       \int_use:N \l_@@_initial_i_int -
4008       \int_use:N \l_@@_initial_j_int
4009     }
4010     {
4011       \int_add:Nn \l_@@_initial_i_int { #3 }
4012       \int_add:Nn \l_@@_initial_j_int { #4 }
4013       \bool_set_true:N \l_@@_initial_open_bool
4014       \bool_set_true:N \l_@@_stop_loop_bool
4015     }
4016     {
4017       \cs_if_exist:cTF
4018       {
4019         pgf @ sh @ ns @ \@@_env:
4020         - \int_use:N \l_@@_initial_i_int
4021         - \int_use:N \l_@@_initial_j_int
4022       }
4023       { \bool_set_true:N \l_@@_stop_loop_bool }
4024       {
4025         \cs_set:cpn
4026         {
4027           @@ _ dotted _
4028           \int_use:N \l_@@_initial_i_int -
4029           \int_use:N \l_@@_initial_j_int
4030         }
4031         { }

```

```

4032         }
4033     }
4034 }
4035 }

```

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.

```

4036 \seq_gput_right:Nx \g_@@_pos_of_xdots_seq
4037 {
4038     { \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`.

```

4039     { \int_min:nn \l_@@_initial_j_int \l_@@_final_j_int }
4040     { \int_use:N \l_@@_final_i_int }
4041     { \int_max:nn \l_@@_initial_j_int \l_@@_final_j_int }
4042     { } % for the name of the block
4043 }
4044 }

```

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

```

4045 \cs_new_protected:Npn \@@_open_shorten:
4046 {
4047     \bool_if:NT \l_@@_initial_open_bool
4048         { \dim_zero:N \l_@@_xdots_shorten_start_dim }
4049     \bool_if:NT \l_@@_final_open_bool
4050         { \dim_zero:N \l_@@_xdots_shorten_end_dim }
4051 }

```

The following command (*when it will be written*) will set the four counters `\l_@@_row_min_int`, `\l_@@_row_max_int`, `\l_@@_col_min_int` and `\l_@@_col_max_int` to the intersections of the submatrices which contains the cell of row #1 and column #2. As of now, it’s only the whole array (excepted exterior rows and columns).

```

4052 \cs_new_protected:Npn \@@_adjust_to_submatrix:nn #1 #2
4053 {
4054     \int_set:Nn \l_@@_row_min_int 1
4055     \int_set:Nn \l_@@_col_min_int 1
4056     \int_set_eq:NN \l_@@_row_max_int \c@iRow
4057     \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`.

```

4058 \seq_map_inline:Nn \g_@@_submatrix_seq
4059 { \@@_adjust_to_submatrix:nnnnn { #1 } { #2 } ##1 }
4060 }

```

#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.

```

4061 \cs_set_protected:Npn \@@_adjust_to_submatrix:nnnnnn #1 #2 #3 #4 #5 #6
4062 {
4063     \bool_if:nT
4064     {
4065         \int_compare_p:n { #3 <= #1 }
4066         && \int_compare_p:n { #1 <= #5 }
4067         && \int_compare_p:n { #4 <= #2 }
4068         && \int_compare_p:n { #2 <= #6 }
4069     }

```

```

4070     {
4071         \int_set:Nn \l_@@_row_min_int { \int_max:nn \l_@@_row_min_int { #3 } }
4072         \int_set:Nn \l_@@_col_min_int { \int_max:nn \l_@@_col_min_int { #4 } }
4073         \int_set:Nn \l_@@_row_max_int { \int_min:nn \l_@@_row_max_int { #5 } }
4074         \int_set:Nn \l_@@_col_max_int { \int_min:nn \l_@@_col_max_int { #6 } }
4075     }
4076 }

4077 \cs_new_protected:Npn \@@_set_initial_coords:
4078 {
4079     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4080     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4081 }
4082 \cs_new_protected:Npn \@@_set_final_coords:
4083 {
4084     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4085     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4086 }
4087 \cs_new_protected:Npn \@@_set_initial_coords_from_anchor:n #1
4088 {
4089     \pgfpointanchor
4090     {
4091         \@@_env:
4092         - \int_use:N \l_@@_initial_i_int
4093         - \int_use:N \l_@@_initial_j_int
4094     }
4095     { #1 }
4096     \@@_set_initial_coords:
4097 }
4098 \cs_new_protected:Npn \@@_set_final_coords_from_anchor:n #1
4099 {
4100     \pgfpointanchor
4101     {
4102         \@@_env:
4103         - \int_use:N \l_@@_final_i_int
4104         - \int_use:N \l_@@_final_j_int
4105     }
4106     { #1 }
4107     \@@_set_final_coords:
4108 }
4109 \cs_new_protected:Npn \@@_open_x_initial_dim:
4110 {
4111     \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
4112     \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
4113     {
4114         \cs_if_exist:cT
4115         { \pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4116         {
4117             \pgfpointanchor
4118             { \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4119             { west }
4120             \dim_set:Nn \l_@@_x_initial_dim
4121             { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
4122         }
4123     }

```

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

```

4124     \dim_compare:nNnT \l_@@_x_initial_dim = \c_max_dim
4125     {
4126         \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4127         \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4128         \dim_add:Nn \l_@@_x_initial_dim \col@sep
4129     }
4130 }

```

```

4131 \cs_new_protected:Npn \@@_open_x_final_dim:
4132 {
4133   \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
4134   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
4135   {
4136     \cs_if_exist:cT
4137     { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4138     {
4139       \pgfpointanchor
4140       { \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4141       { east }
4142       \dim_set:Nn \l_@@_x_final_dim
4143       { \dim_max:nn \l_@@_x_final_dim \pgf@x }
4144     }
4145   }

```

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

```

4146   \dim_compare:nNnT \l_@@_x_final_dim = { - \c_max_dim }
4147   {
4148     \@@_qpoint:n { col - \int_eval:n { \l_@@_final_j_int + 1 } }
4149     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4150     \dim_sub:Nn \l_@@_x_final_dim \col@sep
4151   }
4152 }

```

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.

```

4153 \cs_new_protected:Npn \@@_draw_Ldots:nnn #1 #2 #3
4154 {
4155   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4156   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4157   {
4158     \@@_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.

```

4159   \group_begin:
4160   \@@_open_shorten:
4161   \int_compare:nNnTF { #1 } = 0
4162   { \color { nicematrix-first-row } }
4163   {

```

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.

```

4164     \int_compare:nNnT { #1 } = \l_@@_last_row_int
4165     { \color { nicematrix-last-row } }
4166   }
4167   \keys_set:nn { NiceMatrix / xdots } { #3 }
4168   \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4169   \@@_actually_draw_Ldots:
4170   \group_end:
4171 }
4172 }

```

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`.

```

4173 \cs_new_protected:Npn \@@_actually_draw_Ldots:
4174 {
4175   \bool_if:NTF \l_@@_initial_open_bool
4176   {
4177     \@@_open_x_initial_dim:
4178     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4179     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4180   }
4181   { \@@_set_initial_coords_from_anchor:n { base-east } }
4182   \bool_if:NTF \l_@@_final_open_bool
4183   {
4184     \@@_open_x_final_dim:
4185     \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4186     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4187   }
4188   { \@@_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.

```

4189   \bool_lazy_all:nTF
4190   {
4191     \l_@@_initial_open_bool
4192     \l_@@_final_open_bool
4193     { \int_compare_p:nNn \l_@@_initial_i_int = \l_@@_last_row_int }
4194   }
4195   {
4196     \dim_add:Nn \l_@@_y_initial_dim \c_@@_shift_Ldots_last_row_dim
4197     \dim_add:Nn \l_@@_y_final_dim \c_@@_shift_Ldots_last_row_dim
4198   }

```

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 (?).

```

4199   {
4200     \dim_add:Nn \l_@@_y_initial_dim \l_@@_xdots_radius_dim
4201     \dim_add:Nn \l_@@_y_final_dim \l_@@_xdots_radius_dim
4202   }
4203   \@@_draw_line:
4204 }

```

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.

```

4205 \cs_new_protected:Npn \@@_draw_Cdots:nnn #1 #2 #3
4206 {
4207   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4208   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4209   {
4210     \@@_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.

```

4211   \group_begin:
4212   \@@_open_shorten:
4213   \int_compare:nNnTF { #1 } = 0
4214   { \color { nicematrix-first-row } }
4215   {

```

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.

```

4216         \int_compare:nNnT { #1 } = \l_@@_last_row_int
4217         { \color { nicematrix-last-row } }
4218     }
4219     \keys_set:nn { NiceMatrix / xdots } { #3 }
4220     \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4221     \@@_actually_draw_Cdots:
4222 \group_end:
4223 }
4224 }
```

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`.

```

4225 \cs_new_protected:Npn \@@_actually_draw_Cdots:
4226 {
4227     \bool_if:NTF \l_@@_initial_open_bool
4228     { \@@_open_x_initial_dim: }
4229     { \@@_set_initial_coords_from_anchor:n { mid-east } }
4230     \bool_if:NTF \l_@@_final_open_bool
4231     { \@@_open_x_final_dim: }
4232     { \@@_set_final_coords_from_anchor:n { mid-west } }
4233     \bool_lazy_and:nnTF
4234     \l_@@_initial_open_bool
4235     \l_@@_final_open_bool
4236     {
4237         \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int }
4238         \dim_set_eq:NN \l_tmpa_dim \pgf@y
4239         \@@_qpoint:n { row - \int_eval:n { \l_@@_initial_i_int + 1 } }
4240         \dim_set:Nn \l_@@_y_initial_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
4241         \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
4242     }
4243     {
4244         \bool_if:NT \l_@@_initial_open_bool
4245         { \dim_set_eq:NN \l_@@_y_initial_dim \l_@@_y_final_dim }
4246         \bool_if:NT \l_@@_final_open_bool
4247         { \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim }
4248     }
4249     \@@_draw_line:
4250 }
4251 \cs_new_protected:Npn \@@_open_y_initial_dim:
4252 {
4253     \dim_set:Nn \l_@@_y_initial_dim { - \c_max_dim }
4254     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
4255     {
4256         \cs_if_exist:cT
4257         { pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4258         {
4259             \pgfpointanchor
4260             { \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4261             { north }
4262             \dim_set:Nn \l_@@_y_initial_dim
```

```

4263         { \dim_max:nn \l_@@_y_initial_dim \pgf@y }
4264     }
4265 }
4266 % modified 2023-08-10
4267 \dim_compare:nNnT \l_@@_y_initial_dim = { - \c_max_dim }
4268 {
4269     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4270     \dim_set:Nn \l_@@_y_initial_dim
4271     {
4272         \fp_to_dim:n
4273         {
4274             \pgf@y
4275             + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
4276         }
4277     }
4278 }
4279 }
4280 \cs_new_protected:Npn \@@_open_y_final_dim:
4281 {
4282     \dim_set_eq:NN \l_@@_y_final_dim \c_max_dim
4283     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
4284     {
4285         \cs_if_exist:cT
4286         { pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4287         {
4288             \pgfpointanchor
4289             { \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4290             { south }
4291             \dim_set:Nn \l_@@_y_final_dim
4292             { \dim_min:nn \l_@@_y_final_dim \pgf@y }
4293         }
4294     }
4295 % modified 2023-08-10
4296 \dim_compare:nNnT \l_@@_y_final_dim = \c_max_dim
4297 {
4298     \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4299     \dim_set:Nn \l_@@_y_final_dim
4300     { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
4301 }
4302 }

```

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.

```

4303 \cs_new_protected:Npn \@@_draw_Vdots:nnn #1 #2 #3
4304 {
4305     \@@_adjust_to_submatrix:nn { #1 } { #2 }
4306     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4307     {
4308         \@@_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.

```

4309     \group_begin:
4310     \@@_open_shorten:
4311     \int_compare:nNnTF { #2 } = 0
4312     { \color { nicematrix-first-col } }
4313     {
4314         \int_compare:nNnT { #2 } = \l_@@_last_col_int
4315         { \color { nicematrix-last-col } }
4316     }
4317     \keys_set:nn { NiceMatrix / xdots } { #3 }
4318     \tl_if_empty:VF \l_@@_xdots_color_tl
4319     { \color { \l_@@_xdots_color_tl } }
4320     \@@_actually_draw_Vdots:

```

```

4321     \group_end:
4322   }
4323 }

```

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`.

```

4324 \cs_new_protected:Npn \@@_actually_draw_Vdots:
4325 {

```

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

```

4326   \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.

```

4327   {
4328     \@@_open_y_initial_dim:
4329     \@@_open_y_final_dim:
4330     \int_compare:nNnTF \l_@@_initial_j_int = \c_zero_int

```

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

```

4331     {
4332       \@@_qpoint:n { col - 1 }
4333       \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4334       \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
4335       \dim_sub:Nn \l_@@_x_initial_dim \l_@@_extra_left_margin_dim
4336       % \bool_if:NT \g_@@_delims_bool
4337       % {
4338         \dim_sub:Nn \l_@@_x_initial_dim \c_@@_shift_exterior_Vdots_dim
4339       % }
4340     }
4341     {
4342       \bool_lazy_and:nnTF
4343       { \int_compare_p:nNn \l_@@_last_col_int > { -2 } }
4344       { \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”.

```

4345     {
4346       \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4347       \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4348       \dim_add:Nn \l_@@_x_initial_dim \l_@@_right_margin_dim
4349       \dim_add:Nn \l_@@_x_initial_dim \l_@@_extra_right_margin_dim
4350       % \bool_if:NT \g_@@_delims_bool
4351       % {
4352         \dim_add:Nn
4353         \l_@@_x_initial_dim
4354         \c_@@_shift_exterior_Vdots_dim
4355       % }
4356     }

```

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

```

4357     {
4358       \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4359       \dim_set_eq:NN \l_tmpa_dim \pgf@x
4360       \@@_qpoint:n { col - \int_eval:n { \l_@@_initial_j_int + 1 } }

```

```

4361         \dim_set:Nn \l_@@_x_initial_dim { ( \pgf@x + \l_tmpa_dim ) / 2 }
4362     }
4363 }
4364 }

```

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.

```

4365 {
4366     \bool_set_false:N \l_tmpa_bool
4367     \bool_lazy_and:nnT
4368     { ! \l_@@_initial_open_bool }
4369     { ! \l_@@_final_open_bool }
4370     {
4371         \@@_set_initial_coords_from_anchor:n { south-west }
4372         \@@_set_final_coords_from_anchor:n { north-west }
4373         \bool_set:Nn \l_tmpa_bool
4374         { \dim_compare_p:nNn \l_@@_x_initial_dim = \l_@@_x_final_dim }
4375     }

```

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

```

4376     \bool_if:NTF \l_@@_initial_open_bool
4377     {
4378         \@@_open_y_initial_dim:
4379         \@@_set_final_coords_from_anchor:n { north }
4380         \dim_set_eq:NN \l_@@_x_initial_dim \l_@@_x_final_dim
4381     }
4382     {
4383         \@@_set_initial_coords_from_anchor:n { south }
4384         \bool_if:NTF \l_@@_final_open_bool
4385         \@@_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.

```

4386     {
4387         \@@_set_final_coords_from_anchor:n { north }
4388         \dim_compare:nNnF \l_@@_x_initial_dim = \l_@@_x_final_dim
4389         {
4390             \dim_set:Nn \l_@@_x_initial_dim
4391             {
4392                 \bool_if:NTF \l_tmpa_bool \dim_min:nn \dim_max:nn
4393                 \l_@@_x_initial_dim \l_@@_x_final_dim
4394             }
4395         }
4396     }
4397 }
4398 }
4399 \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
4400 \@@_draw_line:
4401 }

```

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.

```

4402 \cs_new_protected:Npn \@@_draw_Ddots:nnn #1 #2 #3
4403 {
4404     \@@_adjust_to_submatrix:nn { #1 } { #2 }
4405     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4406     {
4407         \@@_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.

```

4408     \group_begin:
4409     \@@_open_shorten:
4410     \keys_set:nn { NiceMatrix / xdots } { #3 }
4411     \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4412     \@@_actually_draw_Ddots:
4413 \group_end:
4414 }
4415 }

```

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.`

```

4416 \cs_new_protected:Npn \@@_actually_draw_Ddots:
4417 {
4418     \bool_if:NTF \l_@@_initial_open_bool
4419     {
4420         \@@_open_y_initial_dim:
4421         \@@_open_x_initial_dim:
4422     }
4423     { \@@_set_initial_coords_from_anchor:n { south-east } }
4424     \bool_if:NTF \l_@@_final_open_bool
4425     {
4426         \@@_open_x_final_dim:
4427         \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4428     }
4429     { \@@_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.

```

4430     \bool_if:NT \l_@@_parallelize_diags_bool
4431     {
4432         \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).

```

4433     \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.

```

4434     {
4435         \dim_gset:Nn \g_@@_delta_x_one_dim
4436         { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4437         \dim_gset:Nn \g_@@_delta_y_one_dim
4438         { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4439     }

```

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`.

```

4440     {
4441         \dim_set:Nn \l_@@_y_final_dim
4442         {
4443             \l_@@_y_initial_dim +

```

```

4444         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4445         \dim_ratio:nn \g_@@_delta_y_one_dim \g_@@_delta_x_one_dim
4446     }
4447 }
4448 }
4449 \@@_draw_line:
4450 }

```

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.

```

4451 \cs_new_protected:Npn \@@_draw_Iddots:nnn #1 #2 #3
4452 {
4453     \@@_adjust_to_submatrix:nn { #1 } { #2 }
4454     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4455     {
4456         \@@_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.

```

4457     \group_begin:
4458     \@@_open_shorten:
4459     \keys_set:nn { NiceMatrix / xdots } { #3 }
4460     \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4461     \@@_actually_draw_Iddots:
4462     \group_end:
4463 }
4464 }

```

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`.

```

4465 \cs_new_protected:Npn \@@_actually_draw_Iddots:
4466 {
4467     \bool_if:NTF \l_@@_initial_open_bool
4468     {
4469         \@@_open_y_initial_dim:
4470         \@@_open_x_initial_dim:
4471     }
4472     { \@@_set_initial_coords_from_anchor:n { south-west } }
4473     \bool_if:NTF \l_@@_final_open_bool
4474     {
4475         \@@_open_y_final_dim:
4476         \@@_open_x_final_dim:
4477     }
4478     { \@@_set_final_coords_from_anchor:n { north-east } }
4479     \bool_if:NT \l_@@_parallelize_diags_bool
4480     {
4481         \int_gincr:N \g_@@_iddots_int
4482         \int_compare:nNnTF \g_@@_iddots_int = 1
4483         {
4484             \dim_gset:Nn \g_@@_delta_x_two_dim
4485             { \l_@@_x_final_dim - \l_@@_x_initial_dim }

```

```

4486         \dim_gset:Nn \g_@@_delta_y_two_dim
4487         { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4488     }
4489     {
4490         \dim_set:Nn \l_@@_y_final_dim
4491         {
4492             \l_@@_y_initial_dim +
4493             ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4494             \dim_ratio:nn \g_@@_delta_y_two_dim \g_@@_delta_x_two_dim
4495         }
4496     }
4497 }
4498 \@@_draw_line:
4499 }

```

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`

```

4500 \cs_new_protected:Npn \@@_draw_line:
4501 {
4502     \pgfrememberpicturerepositiononpagetrue
4503     \pgf@relevantforpicturesizefalse
4504     \bool_lazy_or:nnTF
4505     { \tl_if_eq_p:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl }
4506     \l_@@_dotted_bool
4507     \@@_draw_standard_dotted_line:
4508     \@@_draw_unstandard_dotted_line:
4509 }

```

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.

```

4510 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:
4511 {
4512     \begin { scope }
4513     \@@_draw_unstandard_dotted_line:o
4514     { \l_@@_xdots_line_style_tl , \l_@@_xdots_color_tl }
4515 }

```

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.

```

4516 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:n #1
4517 {
4518     \@@_draw_unstandard_dotted_line:nVWV
4519     { #1 }

```

```

4520 \l_@@_xdots_up_tl
4521 \l_@@_xdots_down_tl
4522 \l_@@_xdots_middle_tl
4523 }
4524 \cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:n { o }

```

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

```

4525 \hook_gput_code:nnn { begindocument } { . }
4526 {
4527   \IfPackageLoadedTF { tikz }
4528   {
4529     \tikzset
4530     {
4531       @@_node_above / .style = { sloped , above } ,
4532       @@_node_below / .style = { sloped , below } ,
4533       @@_node_middle / .style =
4534       {
4535         sloped ,
4536         inner~sep = \c_@@_innersep_middle_dim
4537       }
4538     }
4539   }
4540   { }
4541 }

4542 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:nnnn #1 #2 #3 #4
4543 {

```

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.

```

4544 \dim_zero_new:N \l_@@_l_dim
4545 \dim_set:Nn \l_@@_l_dim
4546 {
4547   \fp_to_dim:n
4548   {
4549     sqrt
4550     (
4551       ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
4552       +
4553       ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
4554     )
4555   }
4556 }
4557 \bool_lazy_and:nnT % security
4558 { \dim_compare_p:nNn { \dim_abs:n \l_@@_l_dim } < \c_@@_max_l_dim }
4559 { \dim_compare_p:nNn { \dim_abs:n \l_@@_l_dim } > { 1 pt } }
4560 {
4561   \dim_set:Nn \l_tmpa_dim
4562   {
4563     \l_@@_x_initial_dim
4564     + ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
4565     * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
4566   }
4567   \dim_set:Nn \l_tmpb_dim
4568   {
4569     \l_@@_y_initial_dim
4570     + ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
4571     * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim

```

```

4572     }
4573     \dim_set:Nn \l_@@_tmpc_dim
4574     {
4575         \l_@@_x_final_dim
4576         - ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
4577         * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
4578     }
4579     \dim_set:Nn \l_@@_tmpd_dim
4580     {
4581         \l_@@_y_final_dim
4582         - ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
4583         * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
4584     }
4585     \dim_set_eq:NN \l_@@_x_initial_dim \l_tmpa_dim
4586     \dim_set_eq:NN \l_@@_y_initial_dim \l_tmpb_dim
4587     \dim_set_eq:NN \l_@@_x_final_dim \l_@@_tmpc_dim
4588     \dim_set_eq:NN \l_@@_y_final_dim \l_@@_tmpd_dim
4589 }

```

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

```

4590     \bool_if:NT \l_@@_xdots_h_labels_bool
4591     {
4592         \tikzset
4593         {
4594             @@_node_above / .style = { auto = left } ,
4595             @@_node_below / .style = { auto = right } ,
4596             @@_node_middle / .style = { inner~sep = \c_@@_innersep_middle_dim }
4597         }
4598     }
4599     \tl_if_empty:nF { #4 }
4600     { \tikzset { @@_node_middle / .append~style = { fill = white } } }
4601     \draw
4602     [ #1 ]
4603     ( \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).

```

4604         -- node [ @@_node_middle ] { $ \scriptstyle #4 $ }
4605         node [ @@_node_below ] { $ \scriptstyle #3 $ }
4606         node [ @@_node_above ] { $ \scriptstyle #2 $ }
4607         ( \l_@@_x_final_dim , \l_@@_y_final_dim ) ;
4608     \end { scope }
4609 }
4610 \cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:nnnn { 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).

```

4611 \cs_new_protected:Npn \@@_draw_standard_dotted_line:
4612 {
4613     \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.

```

4614     \dim_zero_new:N \l_@@_l_dim
4615     \dim_set:Nn \l_@@_l_dim
4616     {
4617         \fp_to_dim:n
4618         {
4619             sqrt
4620             (
4621                 ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
4622                 +
4623                 ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
4624             )

```

```

4625     }
4626 }

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.

4627 \bool_lazy_or:nnF
4628 { \dim_compare_p:nNn { \dim_abs:n \l_@@_l_dim } > \c_@@_max_l_dim }
4629 { \dim_compare_p:nNn \l_@@_l_dim = \c_zero_dim }
4630 \@@_draw_standard_dotted_line_i:
4631 \group_end:
4632 \bool_lazy_all:nF
4633 {
4634   { \tl_if_empty_p:N \l_@@_xdots_up_tl }
4635   { \tl_if_empty_p:N \l_@@_xdots_down_tl }
4636   { \tl_if_empty_p:N \l_@@_xdots_middle_tl }
4637 }
4638 \l_@@_labels_standard_dotted_line:
4639 }
4640 \dim_const:Nn \c_@@_max_l_dim { 50 cm }
4641 \cs_new_protected:Npn \@@_draw_standard_dotted_line_i:
4642 {

```

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

```

4643 \int_set:Nn \l_tmpa_int
4644 {
4645   \dim_ratio:nn
4646   {
4647     \l_@@_l_dim
4648     - \l_@@_xdots_shorten_start_dim
4649     - \l_@@_xdots_shorten_end_dim
4650   }
4651   \l_@@_xdots_inter_dim
4652 }

```

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

```

4653 \dim_set:Nn \l_tmpa_dim
4654 {
4655   ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4656   \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
4657 }
4658 \dim_set:Nn \l_tmpb_dim
4659 {
4660   ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
4661   \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
4662 }

```

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.

```

4663 \dim_gadd:Nn \l_@@_x_initial_dim
4664 {
4665   ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4666   \dim_ratio:nn
4667   {
4668     \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
4669     + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
4670   }
4671   { 2 \l_@@_l_dim }
4672 }
4673 \dim_gadd:Nn \l_@@_y_initial_dim

```

```

4674 {
4675   ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
4676   \dim_ratio:nn
4677   {
4678     \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
4679     + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
4680   }
4681   { 2 \l_@@_l_dim }
4682 }
4683 \pgf@relevantforpicturesizefalse
4684 \int_step_inline:nnn 0 \l_tmpa_int
4685 {
4686   \pgfpathcircle
4687   { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
4688   { \l_@@_xdots_radius_dim }
4689   \dim_add:Nn \l_@@_x_initial_dim \l_tmpa_dim
4690   \dim_add:Nn \l_@@_y_initial_dim \l_tmpb_dim
4691 }
4692 \pgfusepath{fill}
4693 }

4694 \cs_new_protected:Npn \l_@@_labels_standard_dotted_line:
4695 {
4696   \pgfscope
4697   \pgftransformshift
4698   {
4699     \pgfpointlineattime { 0.5 }
4700     { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
4701     { \pgfpoint \l_@@_x_final_dim \l_@@_y_final_dim }
4702   }
4703   \fp_set:Nn \l_tmpa_fp
4704   {
4705     atand
4706     (
4707       \l_@@_y_final_dim - \l_@@_y_initial_dim ,
4708       \l_@@_x_final_dim - \l_@@_x_initial_dim
4709     )
4710   }
4711   \pgftransformrotate { \fp_use:N \l_tmpa_fp }
4712   \bool_if:NF \l_@@_xdots_h_labels_bool { \fp_zero:N \l_tmpa_fp }
4713   \tl_if_empty:NF \l_@@_xdots_middle_tl
4714   {
4715     \begin { pgfscope }
4716     \pgfset { inner~sep = \c_@@_innersep_middle_dim }
4717     \pgfnode
4718     { rectangle }
4719     { center }
4720     {
4721       \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4722       {
4723         \c_math_toggle_token
4724         \scriptstyle \l_@@_xdots_middle_tl
4725         \c_math_toggle_token
4726       }
4727     }
4728     { }
4729     {
4730       \pgfsetfillcolor { white }
4731       \pgfusepath { fill }
4732     }
4733     \end { pgfscope }
4734   }
4735   \tl_if_empty:NF \l_@@_xdots_up_tl

```

```

4736 {
4737   \pgfnode
4738   { rectangle }
4739   { south }
4740   {
4741     \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4742     {
4743       \c_math_toggle_token
4744       \scriptstyle \l_@@_xdots_up_tl
4745       \c_math_toggle_token
4746     }
4747   }
4748   { }
4749   { \pgfusepath { } }
4750 }
4751 \tl_if_empty:NF \l_@@_xdots_down_tl
4752 {
4753   \pgfnode
4754   { rectangle }
4755   { north }
4756   {
4757     \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4758     {
4759       \c_math_toggle_token
4760       \scriptstyle \l_@@_xdots_down_tl
4761       \c_math_toggle_token
4762     }
4763   }
4764   { }
4765   { \pgfusepath { } }
4766 }
4767 \endpgfscope
4768 }

```

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.

```

4769 \hook_gput_code:nnn { begindocument } { . }
4770 {
4771   \tl_set:Nn \l_@@_argspec_tl { m E { _ ^ : } { { } { } { } } }
4772   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
4773   \cs_new_protected:Npn \@@_Ldots
4774   { \@@_collect_options:n { \@@_Ldots_i } }
4775   \exp_args:NNV \NewDocumentCommand \@@_Ldots_i \l_@@_argspec_tl
4776   {
4777     \int_compare:nNnTF \c@jCol = 0
4778     { \@@_error:nn { in-first-col } \Ldots }
4779     {
4780       \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
4781       { \@@_error:nn { in-last-col } \Ldots }
4782       {

```

```

4783         \@@_instruction_of_type:nnn \c_false_bool { Ldots }
4784         { #1 , down = #2 , up = #3 , middle = #4 }
4785     }
4786 }
4787 \bool_if:NF \l_@@_nullify_dots_bool
4788 { \phantom { \ensuremath { \@@_old_ldots } } }
4789 \bool_gset_true:N \g_@@_empty_cell_bool
4790 }

4791 \cs_new_protected:Npn \@@_Cdots
4792 { \@@_collect_options:n { \@@_Cdots_i } }
4793 \exp_args:NNV \NewDocumentCommand \@@_Cdots_i \l_@@_argspec_tl
4794 {
4795     \int_compare:nNnTF \c@jCol = 0
4796     { \@@_error:nn { in~first~col } \Cdots }
4797     {
4798         \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
4799         { \@@_error:nn { in~last~col } \Cdots }
4800         {
4801             \@@_instruction_of_type:nnn \c_false_bool { Cdots }
4802             { #1 , down = #2 , up = #3 , middle = #4 }
4803         }
4804     }
4805     \bool_if:NF \l_@@_nullify_dots_bool
4806     { \phantom { \ensuremath { \@@_old_cdots } } }
4807     \bool_gset_true:N \g_@@_empty_cell_bool
4808 }

4809 \cs_new_protected:Npn \@@_Vdots
4810 { \@@_collect_options:n { \@@_Vdots_i } }
4811 \exp_args:NNV \NewDocumentCommand \@@_Vdots_i \l_@@_argspec_tl
4812 {
4813     \int_compare:nNnTF \c@iRow = 0
4814     { \@@_error:nn { in~first~row } \Vdots }
4815     {
4816         \int_compare:nNnTF \c@iRow = \l_@@_last_row_int
4817         { \@@_error:nn { in~last~row } \Vdots }
4818         {
4819             \@@_instruction_of_type:nnn \c_false_bool { Vdots }
4820             { #1 , down = #2 , up = #3 , middle = #4 }
4821         }
4822     }
4823     \bool_if:NF \l_@@_nullify_dots_bool
4824     { \phantom { \ensuremath { \@@_old_vdots } } }
4825     \bool_gset_true:N \g_@@_empty_cell_bool
4826 }

4827 \cs_new_protected:Npn \@@_Ddots
4828 { \@@_collect_options:n { \@@_Ddots_i } }
4829 \exp_args:NNV \NewDocumentCommand \@@_Ddots_i \l_@@_argspec_tl
4830 {
4831     \int_case:nnF \c@iRow
4832     {
4833         0 { \@@_error:nn { in~first~row } \Ddots }
4834         \l_@@_last_row_int { \@@_error:nn { in~last~row } \Ddots }
4835     }
4836     {
4837         \int_case:nnF \c@jCol
4838         {
4839             0 { \@@_error:nn { in~first~col } \Ddots }
4840             \l_@@_last_col_int { \@@_error:nn { in~last~col } \Ddots }

```

```

4841     }
4842     {
4843       \keys_set_known:nn { NiceMatrix / Ddots } { #1 }
4844       \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Ddots }
4845       { #1 , down = #2 , up = #3 , middle = #4 }
4846     }
4847
4848   }
4849   \bool_if:NF \l_@@_nullify_dots_bool
4850   { \phantom { \ensuremath { \@@_old_ddots } } }
4851   \bool_gset_true:N \g_@@_empty_cell_bool
4852 }

4853 \cs_new_protected:Npn \@@_Iddots
4854 { \@@_collect_options:n { \@@_Iddots_i } }
4855 \exp_args:NNV \NewDocumentCommand \@@_Iddots_i \l_@@_argspec_tl
4856 {
4857   \int_case:nnF \c@iRow
4858   {
4859     0 { \@@_error:nn { in~first~row } \Iddots }
4860     \l_@@_last_row_int { \@@_error:nn { in~last~row } \Iddots }
4861   }
4862   {
4863     \int_case:nnF \c@jCol
4864     {
4865       0 { \@@_error:nn { in~first~col } \Iddots }
4866       \l_@@_last_col_int { \@@_error:nn { in~last~col } \Iddots }
4867     }
4868     {
4869       \keys_set_known:nn { NiceMatrix / Ddots } { #1 }
4870       \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Iddots }
4871       { #1 , down = #2 , up = #3 , middle = #4 }
4872     }
4873   }
4874   \bool_if:NF \l_@@_nullify_dots_bool
4875   { \phantom { \ensuremath { \@@_old_iddots } } }
4876   \bool_gset_true:N \g_@@_empty_cell_bool
4877 }
4878 }

```

End of the \AddToHook.

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

```

4879 \keys_define:nn { NiceMatrix / Ddots }
4880 {
4881   draw-first .bool_set:N = \l_@@_draw_first_bool ,
4882   draw-first .default:n = true ,
4883   draw-first .value_forbidden:n = true
4884 }

```

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

```

4885 \cs_new_protected:Npn \@@_Hspace:
4886 {
4887   \bool_gset_true:N \g_@@_empty_cell_bool
4888   \hspace
4889 }

```

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.

```

4890 \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`.

```

4891 \cs_new:Npn \@@Hdotsfor:
4892 {
4893   \bool_lazy_and:nnTF
4894     { \int_compare_p:nNn \c@jCol = 0 }
4895     { \int_compare_p:nNn \l_@@_first_col_int = 0 }
4896     {
4897       \bool_if:NTF \g_@@_after_col_zero_bool
4898       {
4899         \multicolumn { 1 } { c } { }
4900         \@@Hdotsfor_i
4901       }
4902       { \@@_fatal:n { Hdotsfor~in~col~0 } }
4903     }
4904     {
4905       \multicolumn { 1 } { c } { }
4906       \@@Hdotsfor_i
4907     }
4908 }

```

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:`).

```

4909 \hook_gput_code:nnn { begindocument } { . }
4910 {
4911   \tl_set:Nn \l_@@_argspec_tl { m m 0 { } E { _ ^ : } { { } { } { } } }
4912   \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.

```

4913 \cs_new_protected:Npn \@@Hdotsfor_i
4914 { \@@_collect_options:n { \@@Hdotsfor_ii } }
4915 \exp_args:NNV \NewDocumentCommand \@@Hdotsfor_ii \l_@@_argspec_tl
4916 {
4917   \tl_gput_right:Nx \g_@@_HVdotsfor_lines_tl
4918   {
4919     \@@Hdotsfor:nnnn
4920     { \int_use:N \c@iRow }
4921     { \int_use:N \c@jCol }
4922     { #2 }
4923     {
4924       #1 , #3 ,
4925       down = \exp_not:n { #4 } ,
4926       up = \exp_not:n { #5 } ,
4927       middle = \exp_not:n { #6 }
4928     }
4929   }
4930   \prg_replicate:nn { #2 - 1 }
4931   {
4932     &
4933     \multicolumn { 1 } { c } { }
4934     \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i: % added 2023-08-26
4935   }
4936 }
4937 }

```

```

4938 \cs_new_protected:Npn \@@Hdotsfor:nnnn #1 #2 #3 #4
4939 {
4940   \bool_set_false:N \l_@@_initial_open_bool
4941   \bool_set_false:N \l_@@_final_open_bool

```

For the row, it's easy.

```
4942 \int_set:Nn \l_@@_initial_i_int { #1 }
4943 \int_set_eq:NN \l_@@_final_i_int \l_@@_initial_i_int
```

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

```
4944 \int_compare:nNnTF { #2 } = 1
4945 {
4946   \int_set:Nn \l_@@_initial_j_int 1
4947   \bool_set_true:N \l_@@_initial_open_bool
4948 }
4949 {
4950   \cs_if_exist:cTF
4951   {
4952     pgf @ sh @ ns @ \@@_env:
4953     - \int_use:N \l_@@_initial_i_int
4954     - \int_eval:n { #2 - 1 }
4955   }
4956   { \int_set:Nn \l_@@_initial_j_int { #2 - 1 } }
4957   {
4958     \int_set:Nn \l_@@_initial_j_int { #2 }
4959     \bool_set_true:N \l_@@_initial_open_bool
4960   }
4961 }
4962 \int_compare:nNnTF { #2 + #3 - 1 } = \c@jCol
4963 {
4964   \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
4965   \bool_set_true:N \l_@@_final_open_bool
4966 }
4967 {
4968   \cs_if_exist:cTF
4969   {
4970     pgf @ sh @ ns @ \@@_env:
4971     - \int_use:N \l_@@_final_i_int
4972     - \int_eval:n { #2 + #3 }
4973   }
4974   { \int_set:Nn \l_@@_final_j_int { #2 + #3 } }
4975   {
4976     \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
4977     \bool_set_true:N \l_@@_final_open_bool
4978   }
4979 }
4980 \group_begin:
4981
4982 \@@_open_shorten:
4983
4984
4985 \int_compare:nNnTF { #1 } = 0
4986 { \color { nicematrix-first-row } }
4987 {
4988   \int_compare:nNnT { #1 } = \g_@@_row_total_int
4989   { \color { nicematrix-last-row } }
4990 }
4991
4992 \keys_set:nn { NiceMatrix / xdots } { #4 }
4993 \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4994 \@@_actually_draw_Ldots:
4995 \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).

```
4996 \int_step_inline:nnn { #2 } { #2 + #3 - 1 }
4997 { \cs_set:cpn { @@ _ dotted _ #1 - ##1 } { } }
```

```

4998 }

4999 \hook_gput_code:nnn { begindocument } { . }
5000 {
5001   \tl_set:Nn \l_@@_argspec_tl { m m 0 { } E { _ ^ : } { { } { } { } } }
5002   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
5003   \cs_new_protected:Npn \@@_Vdotsfor:
5004     { \@@_collect_options:n { \@@_Vdotsfor_i } }
5005   \exp_args:NNV \NewDocumentCommand \@@_Vdotsfor_i \l_@@_argspec_tl
5006     {
5007       \bool_gset_true:N \g_@@_empty_cell_bool
5008       \tl_gput_right:Nx \g_@@_HVDotsfor_lines_tl
5009         {
5010           \@@_Vdotsfor:nnnn
5011             { \int_use:N \c@iRow }
5012             { \int_use:N \c@jCol }
5013             { #2 }
5014             {
5015               #1 , #3 ,
5016               down = \exp_not:n { #4 } ,
5017               up = \exp_not:n { #5 } ,
5018               middle = \exp_not:n { #6 }
5019             }
5020           }
5021         }
5022     }

```

```

5023 \cs_new_protected:Npn \@@_Vdotsfor:nnnn #1 #2 #3 #4
5024 {
5025   \bool_set_false:N \l_@@_initial_open_bool
5026   \bool_set_false:N \l_@@_final_open_bool

```

For the column, it's easy.

```

5027   \int_set:Nn \l_@@_initial_j_int { #2 }
5028   \int_set_eq:NN \l_@@_final_j_int \l_@@_initial_j_int

```

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

```

5029   \int_compare:nNnTF { #1 } = 1
5030   {
5031     \int_set:Nn \l_@@_initial_i_int 1
5032     \bool_set_true:N \l_@@_initial_open_bool
5033   }
5034   {
5035     \cs_if_exist:cTF
5036     {
5037       pgf @ sh @ ns @ \@@_env:
5038       - \int_eval:n { #1 - 1 }
5039       - \int_use:N \l_@@_initial_j_int
5040     }
5041     { \int_set:Nn \l_@@_initial_i_int { #1 - 1 } }
5042     {
5043       \int_set:Nn \l_@@_initial_i_int { #1 }
5044       \bool_set_true:N \l_@@_initial_open_bool
5045     }
5046   }
5047   \int_compare:nNnTF { #1 + #3 - 1 } = \c@iRow
5048   {
5049     \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5050     \bool_set_true:N \l_@@_final_open_bool
5051   }
5052   {
5053     \cs_if_exist:cTF
5054     {

```

```

5055         pgf @ sh @ ns @ \@@_env:
5056         - \int_eval:n { #1 + #3 }
5057         - \int_use:N \l_@@_final_j_int
5058     }
5059     { \int_set:Nn \l_@@_final_i_int { #1 + #3 } }
5060     {
5061         \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5062         \bool_set_true:N \l_@@_final_open_bool
5063     }
5064 }
5065 \group_begin:
5066 \@@_open_shorten:
5067
5068
5069
5070
5071 \int_compare:nNnTF { #2 } = 0
5072 { \color { nicematrix-first-col } }
5073 {
5074     \int_compare:nNnT { #2 } = \g_@@_col_total_int
5075     { \color { nicematrix-last-col } }
5076 }
5077 \keys_set:nn { NiceMatrix / xdots } { #4 }
5078 \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
5079 \@@_actually_draw_Vdots:
5080 \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).

```

5081     \int_step_inline:nnn { #1 } { #1 + #3 - 1 }
5082     { \cs_set:cpn { @@ _ dotted _ ##1 - #2 } { } }
5083 }

```

The command `\@@_rotate:` will be linked to `\rotate` in `{NiceArrayWithDelims}`.

```

5084 \NewDocumentCommand \@@_rotate: { 0 { } }
5085 {
5086     \peek_remove_spaces:n
5087     {
5088         \bool_gset_true:N \g_@@_rotate_bool
5089         \keys_set:nn { NiceMatrix / rotate } { #1 }
5090     }
5091 }
5092 \keys_define:nn { NiceMatrix / rotate }
5093 {
5094     c .code:n = \bool_gset_true:N \g_@@_rotate_c_bool ,
5095     c .value_forbidden:n = true ,
5096     unknown .code:n = \@@_error:n { Unknown-key-for-rotate }
5097 }

```

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

```

5098 \cs_new:Npn \@@_double_int_eval:n #1-#2 \q_stop
5099 {
5100   \tl_if_empty:nTF { #2 }
5101     { #1 }
5102     { \@@_double_int_eval_i:n #1-#2 \q_stop }
5103 }
5104 \cs_new:Npn \@@_double_int_eval_i:n #1-#2- \q_stop
5105 { \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).

```

5106 \hook_gput_code:nnn { begindocument } { . }
5107 {
5108   \tl_set:Nn \l_@@_argspec_tl { 0 { } m m ! 0 { } E { _ ^ : } { { } { } { } } }
5109   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
5110   \exp_args:NNV \NewDocumentCommand \@@_line \l_@@_argspec_tl
5111     {
5112       \group_begin:
5113       \keys_set:nn { NiceMatrix / xdots } { #1 , #4 , down = #5 , up = #6 }
5114       \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
5115       \use:e
5116       {
5117         \@@_line_i:nn
5118         { \@@_double_int_eval:n #2 - \q_stop }
5119         { \@@_double_int_eval:n #3 - \q_stop }
5120       }
5121       \group_end:
5122     }
5123 }
5124 \cs_new_protected:Npn \@@_line_i:nn #1 #2
5125 {
5126   \bool_set_false:N \l_@@_initial_open_bool
5127   \bool_set_false:N \l_@@_final_open_bool
5128   \bool_if:nTF
5129     {
5130       \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #1 }
5131       ||
5132       \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #2 }
5133     }
5134     {
5135       \@@_error:nnn { unknown~cell~for~line~in~CodeAfter } { #1 } { #2 }
5136     }

```

The test of `measuring@` is a security (cf. question 686649 on TeX StackExchange).

```

5137   { \legacy_if:nF { measuring@ } { \@@_draw_line_ii:nn { #1 } { #2 } } }
5138 }
5139 \hook_gput_code:nnn { begindocument } { . }
5140 {
5141   \cs_new_protected:Npx \@@_draw_line_ii:nn #1 #2
5142   {

```

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:`.

```

5143     \c_@@_pgfortikzpicture_tl
5144     \@@_draw_line_iii:nn { #1 } { #2 }
5145     \c_@@_endpgfortikzpicture_tl
5146   }
5147 }

```

The following command *must* be protected (it’s used in the construction of `\@@_draw_line_ii:nn`).

```

5148 \cs_new_protected:Npn \@@_draw_line_iii:nn #1 #2
5149 {
5150   \pgfrememberpicturepositiononpagetrue
5151   \pgfpointshapeborder { \@@_env: - #1 } { \@@_qpoint:n { #2 } }
5152   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
5153   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
5154   \pgfpointshapeborder { \@@_env: - #2 } { \@@_qpoint:n { #1 } }
5155   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
5156   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
5157   \@@_draw_line:
5158 }

```

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`

```

5159 \keys_define:nn { NiceMatrix / RowStyle }
5160 {
5161   cell-space-top-limit .dim_set:N = \l_tmpa_dim ,
5162   cell-space-top-limit .initial:n = \c_zero_dim ,
5163   cell-space-top-limit .value_required:n = true ,
5164   cell-space-bottom-limit .dim_set:N = \l_tmpb_dim ,
5165   cell-space-bottom-limit .initial:n = \c_zero_dim ,
5166   cell-space-bottom-limit .value_required:n = true ,
5167   cell-space-limits .meta:n =
5168   {
5169     cell-space-top-limit = #1 ,
5170     cell-space-bottom-limit = #1 ,
5171   } ,
5172   color .tl_set:N = \l_@@_color_tl ,
5173   color .value_required:n = true ,
5174   bold .bool_set:N = \l_tmpa_bool ,
5175   bold .default:n = true ,
5176   bold .initial:n = false ,
5177   nb-rows .code:n =
5178   \str_if_eq:nnTF { #1 } { * }
5179   { \int_set:Nn \l_@@_key_nb_rows_int { 500 } }
5180   { \int_set:Nn \l_@@_key_nb_rows_int { #1 } } ,
5181   nb-rows .value_required:n = true ,
5182   rowcolor .tl_set:N = \l_tmpa_tl ,
5183   rowcolor .value_required:n = true ,
5184   rowcolor .initial:n = ,
5185   unknown .code:n = \@@_error:n { Unknown~key~for~RowStyle }
5186 }

5187 \NewDocumentCommand \@@_RowStyle:n { 0 { } m }
5188 {

```

```

5189 \group_begin:
5190 \tl_clear:N \l_tmpa_tl % value of \rowcolor
5191 \tl_clear:N \l_@@_color_tl
5192 \int_set:Nn \l_@@_key_nb_rows_int 1
5193 \keys_set:nn { NiceMatrix / RowStyle } { #1 }

```

If the key `rowcolor` has been used.

```

5194 \tl_if_empty:NF \l_tmpa_tl
5195 {

```

First, the end of the current row (we remind that `\RowStyle` applies to the *end* of the current row).

```

5196 \tl_gput_right:Nx \g_@@_pre_code_before_tl
5197 {

```

The command `\@@_exp_color_arg:NV` is *fully expandable*.

```

5198 \@@_exp_color_arg:NV \@@_rectanglecolor \l_tmpa_tl
5199 { \int_use:N \c@iRow - \int_use:N \c@jCol }
5200 { \int_use:N \c@iRow - * }
5201 }

```

Then, the other rows (if there is several rows).

```

5202 \int_compare:nNnT \l_@@_key_nb_rows_int > 1
5203 {
5204   \tl_gput_right:Nx \g_@@_pre_code_before_tl
5205   {
5206     \@@_exp_color_arg:NV \@@_rowcolor \l_tmpa_tl
5207     {
5208       \int_eval:n { \c@iRow + 1 }
5209       - \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int - 1 }
5210     }
5211   }
5212 }
5213 }
5214 \tl_gput_right:Nn \g_@@_row_style_tl { \ifnum \c@iRow < }
5215 \tl_gput_right:Nx \g_@@_row_style_tl
5216 { \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int } }
5217 \tl_gput_right:Nn \g_@@_row_style_tl { #2 }

```

`\l_tmpa_dim` is the value of the key `cell-space-top-limit` of `\RowStyle`.

```

5218 \dim_compare:nNnT \l_tmpa_dim > \c_zero_dim
5219 {
5220   \tl_gput_right:Nx \g_@@_row_style_tl
5221   {
5222     \tl_gput_right:Nn \exp_not:N \g_@@_cell_after_hook_tl
5223     {
5224       \dim_set:Nn \l_@@_cell_space_top_limit_dim
5225       { \dim_use:N \l_tmpa_dim }
5226     }
5227   }
5228 }

```

`\l_tmpb_dim` is the value of the key `cell-space-bottom-limit` of `\RowStyle`.

```

5229 \dim_compare:nNnT \l_tmpb_dim > \c_zero_dim
5230 {
5231   \tl_gput_right:Nx \g_@@_row_style_tl
5232   {
5233     \tl_gput_right:Nn \exp_not:N \g_@@_cell_after_hook_tl
5234     {
5235       \dim_set:Nn \l_@@_cell_space_bottom_limit_dim
5236       { \dim_use:N \l_tmpb_dim }
5237     }
5238   }
5239 }

```

`\l_@@_color_tl` is the value of the key `color` of `\RowStyle`.

```

5240 \tl_if_empty:NF \l_@@_color_tl
5241 {
5242   \tl_gput_right:Nx \g_@@_row_style_tl

```

```

5243     {
5244         \mode_leave_vertical:
5245         \@@_color:n { \l_@@_color_tl }
5246     }
5247 }
\l_tmpa_bool is the value of the key bold.
5248 \bool_if:NT \l_tmpa_bool
5249 {
5250     \tl_gput_right:Nn \g_@@_row_style_tl
5251     {
5252         \if_mode_math:
5253             \c_math_toggle_token
5254             \bfseries \boldmath
5255             \c_math_toggle_token
5256         \else:
5257             \bfseries \boldmath
5258         \fi:
5259     }
5260 }
5261 \tl_gput_right:Nn \g_@@_row_style_tl { \fi }
5262 \group_end:
5263 \g_@@_row_style_tl
5264 \ignorespaces
5265 }

```

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).

```

5266 \cs_new_protected:Npn \@@_add_to_colors_seq:nn #1 #2
5267 {

```

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.

```

5268     \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`.

```

5269     \str_if_in:nnF { #1 } { !! }
5270     {
5271         \seq_map_indexed_inline:Nn \g_@@_colors_seq
5272         { \tl_if_eq:nnT { #1 } { ##2 } { \int_set:Nn \l_tmpa_int { ##1 } } }
5273     }
5274     \int_compare:nNnTF \l_tmpa_int = \c_zero_int

```

First, the case where the color is a *new* color (not in the sequence).

```

5275     {
5276         \seq_gput_right:Nn \g_@@_colors_seq { #1 }
5277         \tl_gset:cx { g_@@_color _ \seq_count:N \g_@@_colors_seq _ tl } { #2 }
5278     }

```

Now, the case where the color is *not* a new color (the color is in the sequence at the position `\l_tmpa_int`).

```

5279     { \tl_gput_right:cx { g_@@_color _ \int_use:N \l_tmpa_int _tl } { #2 } }
5280 }

5281 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { x n }
5282 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { x x }

```

The following command must be used within a `\pgfpicture`.

```

5283 \cs_new_protected:Npn \@@_clip_with_rounded_corners:
5284 {
5285     \dim_compare:nNnT \l_@@_tab_rounded_corners_dim > \c_zero_dim
5286     {

```

The TeX group is for `\pgfsetcornersarced` (whose scope is the TeX scope).

```

5287     \group_begin:
5288     \pgfsetcornersarced
5289     {
5290         \pgfpoint
5291         { \l_@@_tab_rounded_corners_dim }
5292         { \l_@@_tab_rounded_corners_dim }
5293     }

```

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.

```

5294     \bool_if:NTF \l_@@_hvlines_bool
5295     {
5296         \pgfpathrectanglecorners
5297         {
5298             \pgfpointadd
5299             { \@@_qpoint:n { row-1 } }
5300             { \pgfpoint { 0.5 \arrayrulewidth } { \c_zero_dim } }
5301         }
5302         {
5303             \pgfpointadd
5304             {
5305                 \@@_qpoint:n
5306                 { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
5307             }
5308             { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
5309         }
5310     }
5311     {
5312         \pgfpathrectanglecorners
5313         { \@@_qpoint:n { row-1 } }
5314         {
5315             \pgfpointadd
5316             {
5317                 \@@_qpoint:n
5318                 { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
5319             }
5320             { \pgfpoint \c_zero_dim \arrayrulewidth }
5321         }
5322     }
5323     \pgfusepath { clip }
5324     \group_end:

```

The TeX group was for `\pgfsetcornersarced`.

```
5325   }
5326 }
```

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`).

```
5327 \cs_new_protected:Npn \@@_actually_color:
5328 {
5329   \pgfpicture
5330   \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.

```
5331   \@@_clip_with_rounded_corners:
5332   \seq_map_indexed_inline:Nn \g_@@_colors_seq
5333   {
5334     \begin { pgfscope }
5335       \@@_color_opacity ##2
5336       \use:c { g_@@_color _ ##1 _tl }
5337       \tl_gclear:c { g_@@_color _ ##1 _tl }
5338       \pgfusepath { fill }
5339     \end { pgfscope }
5340   }
5341   \endpgfpicture
5342 }
```

The following command will extract the potential key `opacity` in its optional argument (between square brackets) and (of course) then apply the command `\color`.

```
5343 \cs_new_protected:Npn \@@_color_opacity
5344 {
5345   \peek_meaning:NTF [
5346     { \@@_color_opacity:w }
5347     { \@@_color_opacity:w [ ] }
5348   }
```

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.

```
5349 \cs_new_protected:Npn \@@_color_opacity:w [ #1 ]
5350 {
5351   \tl_clear:N \l_tmpa_tl
5352   \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.

```
5353   \tl_if_empty:NF \l_tmpa_tl { \exp_args:NV \pgfsetfillopacity \l_tmpa_tl }
5354   \tl_if_empty:NTF \l_tmpb_tl
5355     { \@declaredcolor }
5356     { \use:x { \exp_not:N \@undeclaredcolor [ \l_tmpb_tl ] } }
5357 }
```

The following set of keys is used by the command `\@@_color_opacity:wn`.

```
5358 \keys_define:nn { nicematrix / color-opacity }
5359 {
5360   opacity .tl_set:N      = \l_tmpa_tl ,
5361   opacity .value_required:n = true
5362 }
```

```

5363 \cs_new_protected:Npn \@@_cartesian_color:nn #1 #2
5364 {
5365   \tl_set:Nn \l_@@_rows_tl { #1 }
5366   \tl_set:Nn \l_@@_cols_tl { #2 }
5367   \@@_cartesian_path:
5368 }

```

Here is an example : `\@@_rowcolor {red!15} {1,3,5-7,10-}`

```

5369 \NewDocumentCommand \@@_rowcolor { 0 { } m m }
5370 {
5371   \tl_if_blank:nF { #2 }
5372   {
5373     \@@_add_to_colors_seq:xn
5374     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5375     { \@@_cartesian_color:nn { #3 } { - } }
5376   }
5377 }

```

Here an example : `\@@_columncolor:nn {red!15} {1,3,5-7,10-}`

```

5378 \NewDocumentCommand \@@_columncolor { 0 { } m m }
5379 {
5380   \tl_if_blank:nF { #2 }
5381   {
5382     \@@_add_to_colors_seq:xn
5383     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5384     { \@@_cartesian_color:nn { - } { #3 } }
5385   }
5386 }

```

Here is an example : `\@@_rectanglecolor{red!15}{2-3}{5-6}`

```

5387 \NewDocumentCommand \@@_rectanglecolor { 0 { } m m m }
5388 {
5389   \tl_if_blank:nF { #2 }
5390   {
5391     \@@_add_to_colors_seq:xn
5392     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5393     { \@@_rectanglecolor:nnn { #3 } { #4 } { 0 pt } }
5394   }
5395 }

```

The last argument is the radius of the corners of the rectangle.

```

5396 \NewDocumentCommand \@@_roundedrectanglecolor { 0 { } m m m m }
5397 {
5398   \tl_if_blank:nF { #2 }
5399   {
5400     \@@_add_to_colors_seq:xn
5401     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5402     { \@@_rectanglecolor:nnn { #3 } { #4 } { #5 } }
5403   }
5404 }

```

The last argument is the radius of the corners of the rectangle.

```

5405 \cs_new_protected:Npn \@@_rectanglecolor:nnn #1 #2 #3
5406 {
5407   \@@_cut_on_hyphen:w #1 \q_stop
5408   \tl_clear_new:N \l_@@_tmpc_tl
5409   \tl_clear_new:N \l_@@_tmpd_tl
5410   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
5411   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
5412   \@@_cut_on_hyphen:w #2 \q_stop
5413   \tl_set:Nx \l_@@_rows_tl { \l_@@_tmpc_tl - \l_tmpa_tl }
5414   \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`.

```
5415 \@@_cartesian_path:n { #3 }
5416 }
```

Here is an example: `\@@_cellcolor[rgb]{0.5,0.5,0}{2-3,3-4,4-5,5-6}`

```
5417 \NewDocumentCommand \@@_cellcolor { 0 { } m m }
5418 {
5419   \clist_map_inline:nn { #3 }
5420     { \@@_rectanglecolor [ #1 ] { #2 } { ##1 } { ##1 } }
5421 }
```

```
5422 \NewDocumentCommand \@@_chessboardcolors { 0 { } m m }
5423 {
5424   \int_step_inline:nn { \int_use:N \c@iRow }
5425     {
5426       \int_step_inline:nn { \int_use:N \c@jCol }
5427         {
5428           \int_if_even:nTF { ####1 + ##1 }
5429             { \@@_cellcolor [ #1 ] { #2 } }
5430             { \@@_cellcolor [ #1 ] { #3 } }
5431           { ##1 - ####1 }
5432         }
5433     }
5434 }
```

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”.

```
5435 \NewDocumentCommand \@@_arraycolor { 0 { } m }
5436 {
5437   \@@_rectanglecolor [ #1 ] { #2 }
5438     { 1 - 1 }
5439     { \int_use:N \c@iRow - \int_use:N \c@jCol }
5440 }
```

```
5441 \keys_define:nn { NiceMatrix / rowcolors }
5442 {
5443   respect-blocks .bool_set:N = \l_@@_respect_blocks_bool ,
5444   respect-blocks .default:n = true ,
5445   cols .tl_set:N = \l_@@_cols_tl ,
5446   restart .bool_set:N = \l_@@_rowcolors_restart_bool ,
5447   restart .default:n = true ,
5448   unknown .code:n = \@@_error:n { Unknown~key~for~rowcolors }
5449 }
```

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*.

```
5450 \NewDocumentCommand \@@_rowlistcolors { 0 { } m m 0 { } }
5451 {
```

The group is for the options. `\l_@@_colors_seq` will be the list of colors.

```

5452 \group_begin:
5453 \seq_clear_new:N \l_@@_colors_seq
5454 \seq_set_split:Nnn \l_@@_colors_seq { , } { #3 }
5455 \tl_clear_new:N \l_@@_cols_tl
5456 \tl_set:Nn \l_@@_cols_tl { - }
5457 \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).

```

5458 \int_zero_new:N \l_@@_color_int
5459 \int_set:Nn \l_@@_color_int 1
5460 \bool_if:NT \l_@@_respect_blocks_bool
5461 {

```

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`).

```

5462 \seq_set_eq:NN \l_tmpb_seq \g_@@_pos_of_blocks_seq
5463 \seq_set_filter:Nnn \l_tmpa_seq \l_tmpb_seq
5464 { \@@_not_in_exterior_p:nnnnn ##1 }
5465 }
5466 \pgfpicture
5467 \pgf@relevantforpicturesizefalse

```

`#2` is the list of intervals of rows.

```

5468 \clist_map_inline:nn { #2 }
5469 {
5470 \tl_set:Nn \l_tmpa_tl { ##1 }
5471 \tl_if_in:NnTF \l_tmpa_tl { - }
5472 { \@@_cut_on_hyphen:w ##1 \q_stop }
5473 { \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.

```

5474 \int_set:Nn \l_tmpa_int \l_tmpa_tl
5475 \int_set:Nn \l_@@_color_int
5476 { \bool_if:NTF \l_@@_rowcolors_restart_bool 1 \l_tmpa_tl }
5477 \int_zero_new:N \l_@@_tmpc_int
5478 \int_set:Nn \l_@@_tmpc_int \l_tmpb_tl
5479 \int_do_until:nNnn \l_tmpa_int > \l_@@_tmpc_int
5480 {

```

We will compute in `\l_tmpb_int` the last row of the “block”.

```

5481 \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).

```

5482 \bool_if:NT \l_@@_respect_blocks_bool
5483 {
5484 \seq_set_filter:Nnn \l_tmpb_seq \l_tmpa_seq
5485 { \@@_intersect_our_row_p:nnnnn #####1 }
5486 \seq_map_inline:Nn \l_tmpb_seq { \@@_rowcolors_i:nnnnn #####1 }

```

Now, the last row of the block is computed in `\l_tmpb_int`.

```

5487 }
5488 \tl_set:Nx \l_@@_rows_tl
5489 { \int_use:N \l_tmpa_int - \int_use:N \l_tmpb_int }

```

`\l_@@_tmpc_tl` will be the color that we will use.

```

5490 \tl_clear_new:N \l_@@_color_tl
5491 \tl_set:Nx \l_@@_color_tl
5492 {
5493 \@@_color_index:n
5494 {
5495 \int_mod:nn
5496 { \l_@@_color_int - 1 }
5497 { \seq_count:N \l_@@_colors_seq }

```

```

5498         + 1
5499     }
5500 }
5501 \tl_if_empty:NF \l_@@_color_tl
5502 {
5503     \@@_add_to_colors_seq:xx
5504     { \tl_if_blank:nF { #1 } { [ #1 ] } { \l_@@_color_tl } }
5505     { \@@_cartesian_color:nn { \l_@@_rows_tl } { \l_@@_cols_tl } }
5506 }
5507 \int_incr:N \l_@@_color_int
5508 \int_set:Nn \l_tmpa_int { \l_tmpb_int + 1 }
5509 }
5510 }
5511 \endpgfpicture
5512 \group_end:
5513 }

```

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.

```

5514 \cs_new:Npn \@@_color_index:n #1
5515 {
5516     \str_if_eq:eeTF { \seq_item:Nn \l_@@_colors_seq { #1 } } { = }
5517     { \@@_color_index:n { #1 - 1 } }
5518     { \seq_item:Nn \l_@@_colors_seq { #1 } }
5519 }

```

The command `\rowcolors` (available in the `\CodeBefore`) is a specialisation of the most general command `\rowlistcolors`. The last argument, which is an optional argument between square brackets is provided by curryfication.

```

5520 \NewDocumentCommand \@@_rowcolors { 0 { } m m m }
5521 { \@@_rowlistcolors [ #1 ] { #2 } { { #3 } , { #4 } } }

5522 \cs_new_protected:Npn \@@_rowcolors_i:nnnnn #1 #2 #3 #4 #5
5523 {
5524     \int_compare:nNnT { #3 } > \l_tmpb_int
5525     { \int_set:Nn \l_tmpb_int { #3 } }
5526 }

5527 \prg_new_conditional:Nnn \@@_not_in_exterior:nnnnn p
5528 {
5529     \bool_lazy_or:nnTF
5530     { \int_compare_p:nNn { #4 } = \c_zero_int }
5531     { \int_compare_p:nNn { #2 } = { \int_eval:n { \c@jCol + 1 } } }
5532     \prg_return_false:
5533     \prg_return_true:
5534 }

```

The following command return true when the block intersects the row `\l_tmpa_int`.

```

5535 \prg_new_conditional:Nnn \@@_intersect_our_row:nnnnn p
5536 {
5537     \bool_if:nTF
5538     {
5539         \int_compare_p:n { #1 <= \l_tmpa_int }
5540         &&
5541         \int_compare_p:n { \l_tmpa_int <= #3 }
5542     }
5543     \prg_return_true:
5544     \prg_return_false:
5545 }

```

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`).

```

5546 \cs_new_protected:Npn \@@_cartesian_path:n #1
5547 {
5548   \bool_lazy_and:nnT
5549   { ! \seq_if_empty_p:N \l_@@_corners_cells_seq }
5550   { \dim_compare_p:nNn { #1 } = \c_zero_dim }
5551   {
5552     \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
5553     \@@_expand_clist:NN \l_@@_rows_tl \c@iRow
5554   }

```

We begin the loop over the columns.

```

5555   \clist_map_inline:Nn \l_@@_cols_tl
5556   {
5557     \tl_set:Nn \l_tmpa_tl { ##1 }
5558     \tl_if_in:NnTF \l_tmpa_tl { - }
5559     { \@@_cut_on_hyphen:w ##1 \q_stop }
5560     { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
5561     \bool_lazy_or:nnT
5562     { \tl_if_blank_p:V \l_tmpa_tl }
5563     { \str_if_eq_p:Vn \l_tmpa_tl { * } }
5564     { \tl_set:Nn \l_tmpa_tl { 1 } }
5565     \bool_lazy_or:nnT
5566     { \tl_if_blank_p:V \l_tmpb_tl }
5567     { \str_if_eq_p:Vn \l_tmpb_tl { * } }
5568     { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@jCol } }
5569     \int_compare:nNnT \l_tmpb_tl > \c@jCol
5570     { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@jCol } }

```

`\l_@@_tmpc_tl` will contain the number of column.

```

5571     \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.

```

5572     \@@_qpoint:n { col - \l_tmpa_tl }
5573     \int_compare:nNnTF \l_@@_first_col_int = \l_tmpa_tl
5574     { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
5575     { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
5576     \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
5577     \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }

```

We begin the loop over the rows.

```

5578   \clist_map_inline:Nn \l_@@_rows_tl
5579   {
5580     \tl_set:Nn \l_tmpa_tl { ####1 }
5581     \tl_if_in:NnTF \l_tmpa_tl { - }
5582     { \@@_cut_on_hyphen:w ####1 \q_stop }
5583     { \@@_cut_on_hyphen:w ####1 - ####1 \q_stop }
5584     \tl_if_empty:NT \l_tmpa_tl { \tl_set:Nn \l_tmpa_tl { 1 } }
5585     \tl_if_empty:NT \l_tmpb_tl
5586     { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@iRow } }
5587     \int_compare:nNnT \l_tmpb_tl > \c@iRow
5588     { \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`.

```

5589     \seq_if_in:NxF \l_@@_corners_cells_seq
5590     { \l_tmpa_tl - \l_@@_tmpc_tl }
5591     {
5592       \@@_qpoint:n { row - \int_eval:n { \l_tmpb_tl + 1 } }

```

```

5593         \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
5594         \@@_qpoint:n { row - \l_tmpa_tl }
5595         \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
5596         \pgfsetcornersarced { \pgfpoint { #1 } { #1 } }
5597         \pgfpathrectanglecorners
5598         { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
5599         { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5600     }
5601 }
5602 }
5603 }

```

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`).

```

5604 \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.

```

5605 \cs_new_protected:Npn \@@_expand_clist:NN #1 #2
5606 {
5607     \clist_set_eq:NN \l_tmpa_clist #1
5608     \clist_clear:N #1
5609     \clist_map_inline:Nn \l_tmpa_clist
5610     {
5611         \tl_set:Nn \l_tmpa_tl { ##1 }
5612         \tl_if_in:NnTF \l_tmpa_tl { - }
5613         { \@@_cut_on_hyphen:w ##1 \q_stop }
5614         { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
5615         \bool_lazy_or:nnT
5616         { \tl_if_blank_p:V \l_tmpa_tl }
5617         { \str_if_eq_p:Vn \l_tmpa_tl { * } }
5618         { \tl_set:Nn \l_tmpa_tl { 1 } }
5619         \bool_lazy_or:nnT
5620         { \tl_if_blank_p:V \l_tmpb_tl }
5621         { \str_if_eq_p:Vn \l_tmpb_tl { * } }
5622         { \tl_set:Nx \l_tmpb_tl { \int_use:N #2 } }
5623         \int_compare:nNnT \l_tmpb_tl > #2
5624         { \tl_set:Nx \l_tmpb_tl { \int_use:N #2 } }
5625         \int_step_inline:nnn \l_tmpa_tl \l_tmpb_tl
5626         { \clist_put_right:Nn #1 { #####1 } }
5627     }
5628 }

```

When the user uses the key `color-inside`, the following command will be linked to `\cellcolor` in the tabular.

```

5629 \NewDocumentCommand \@@_cellcolor_tabular { 0 { } m }
5630 {
5631     \tl_gput_right:Nx \g_@@_pre_code_before_tl
5632     {

```

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`).

```

5633         \@@_cellcolor [ #1 ] { \exp_not:n { #2 } }
5634         { \int_use:N \c@iRow - \int_use:N \c@jCol }
5635     }
5636     \ignorespaces
5637 }

```

When the user uses the key `color-inside`, the following command will be linked to `\rowcolor` in the tabular.

```

5638 \NewDocumentCommand \@@_rowcolor_tabular { 0 { } m }
5639 {
5640   \tl_gput_right:Nx \g_@@_pre_code_before_tl
5641   {
5642     \@@_rectanglecolor [ #1 ] { \exp_not:n { #2 } }
5643     { \int_use:N \c@iRow - \int_use:N \c@jCol }
5644     { \int_use:N \c@iRow - \exp_not:n { \int_use:N \c@jCol } }
5645   }
5646   \ignorespaces
5647 }

```

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).

```

5648 \NewDocumentCommand { \@@_rowcolors_tabular } { 0 { } m m }
5649 { \@@_rowlistcolors_tabular [ #1 ] { #2 , #3 } }

```

When the user uses the key `color-inside`, the following command will be linked to `\rowlistcolors` in the tabular.

```

5650 \NewDocumentCommand { \@@_rowlistcolors_tabular } { 0 { } m 0 { } }
5651 {
5652   \peek_remove_spaces:n
5653   {
5654     \tl_gput_right:Nx \g_@@_nicematrix_pre_code_before_tl
5655     {
5656       \@@_rowlistcolors
5657       [ #1 ] { \int_use:N \c@iRow } { #2 }
5658       [ restart, cols = \int_use:N \c@jCol - , #3 ]
5659     }
5660   }
5661 }

```

```

5662 \NewDocumentCommand \@@_columncolor_preamble { 0 { } m }
5663 {

```

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).

```

5664   \int_compare:nNnT \c@jCol > \g_@@_col_total_int
5665   {

```

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).

```

5666     \tl_gput_left:Nx \g_@@_pre_code_before_tl
5667     {
5668       \exp_not:N \columncolor [ #1 ]
5669       { \exp_not:n { #2 } } { \int_use:N \c@jCol }
5670     }
5671   }
5672 }

```

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`.

```
5673 \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`.

```
5674 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix:n #1
5675 {
5676   \int_compare:nNnTF \l_@@_first_col_int = 0
5677   { \@@_OnlyMainNiceMatrix_i:n { #1 } }
5678   {
5679     \int_compare:nNnTF \c@jCol = 0
5680     {
5681       \int_compare:nNnF \c@iRow = { -1 }
5682       { \int_compare:nNnF \c@iRow = { \l_@@_last_row_int - 1 } { #1 } }
5683     }
5684     { \@@_OnlyMainNiceMatrix_i:n { #1 } }
5685   }
5686 }
```

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.

```
5687 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix_i:n #1
5688 {
5689   \int_compare:nNnF \c@iRow = 0
5690   {
5691     \int_compare:nNnF \c@iRow = \l_@@_last_row_int
5692     {
5693       \int_compare:nNnT \c@jCol > \c_zero_int
5694       { \bool_if:NF \l_@@_in_last_col_bool { #1 } }
5695     }
5696   }
5697 }
```

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.

```
5698 \keys_define:nn { NiceMatrix / Rules }
5699 {
5700   position .int_set:N = \l_@@_position_int ,
5701   position .value_required:n = true ,
5702   start .int_set:N = \l_@@_start_int ,
5703   start .initial:n = 1 ,
5704   end .code:n =
5705     \bool_lazy_or:nnTF
5706     { \tl_if_empty_p:n { #1 } }
5707     { \str_if_eq_p:nn { #1 } { last } }
5708     { \int_set_eq:NN \l_@@_end_int \c@jCol }
5709     { \int_set:Nn \l_@@_end_int { #1 } }
5710 }
```

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.

```

5711 \keys_define:nn { NiceMatrix / RulesBis }
5712 {
5713   multiplicity .int_set:N = \l_@@_multiplicity_int ,
5714   multiplicity .initial:n = 1 ,
5715   dotted .bool_set:N = \l_@@_dotted_bool ,
5716   dotted .initial:n = false ,
5717   dotted .default:n = true ,
5718   color .code:n = \@@_set_CT@arc@:n { #1 } ,
5719   color .value_required:n = true ,
5720   sep-color .code:n = \@@_set_CT@drsc@:n { #1 } ,
5721   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.

```

5722   tikz .tl_set:N = \l_@@_tikz_rule_tl ,
5723   tikz .value_required:n = true ,
5724   tikz .initial:n = ,
5725   total-width .dim_set:N = \l_@@_rule_width_dim ,
5726   total-width .value_required:n = true ,
5727   width .meta:n = { total-width = #1 } ,
5728   unknown .code:n = \@@_error:n { Unknown-key-for-RulesBis }
5729 }

```

The vertical rules

The following command will be executed in the internal `\CodeAfter`. The argument `#1` is a list of `key=value` pairs.

```

5730 \cs_new_protected:Npn \@@_vline:n #1
5731 {

```

The group is for the options.

```

5732   \group_begin:
5733   \int_zero_new:N \l_@@_end_int
5734   \int_set_eq:NN \l_@@_end_int \c@iRow
5735   \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).

```

5736   \int_compare:nNnT \l_@@_position_int < { \c@jCol + 2 }
5737   \@@_vline_i:
5738   \group_end:
5739 }

```

```

5740 \cs_new_protected:Npn \@@_vline_i:
5741 {
5742   \int_zero_new:N \l_@@_local_start_int
5743   \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`.

```

5744   \tl_set:Nx \l_tmpb_tl { \int_eval:n \l_@@_position_int }
5745   \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
5746   \l_tmpa_tl
5747   {

```

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.

```

5748 \bool_gset_true:N \g_tmpa_bool
5749 \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
5750 { \@@_test_vline_in_block:nnnnn ##1 }
5751 \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
5752 { \@@_test_vline_in_block:nnnnn ##1 }
5753 \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
5754 { \@@_test_vline_in_stroken_block:nnnn ##1 }
5755 \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_v:
5756 \bool_if:NTF \g_tmpa_bool
5757 {
5758   \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.

```

5759   { \int_set:Nn \l_@@_local_start_int \l_tmpa_tl }
5760 }
5761 {
5762   \int_compare:nNnT \l_@@_local_start_int > 0
5763   {
5764     \int_set:Nn \l_@@_local_end_int { \l_tmpa_tl - 1 }
5765     \@@_vline_ii:
5766     \int_zero:N \l_@@_local_start_int
5767   }
5768 }
5769 }
5770 \int_compare:nNnT \l_@@_local_start_int > 0
5771 {
5772   \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
5773   \@@_vline_ii:
5774 }
5775 }

```

```

5776 \cs_new_protected:Npn \@@_test_in_corner_v:
5777 {
5778   \int_compare:nNnTF \l_tmpb_tl = { \int_eval:n { \c@jCol + 1 } }
5779   {
5780     \seq_if_in:NxT
5781     \l_@@_corners_cells_seq
5782     { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
5783     { \bool_set_false:N \g_tmpa_bool }
5784   }
5785   {
5786     \seq_if_in:NxT
5787     \l_@@_corners_cells_seq
5788     { \l_tmpa_tl - \l_tmpb_tl }
5789     {
5790       \int_compare:nNnTF \l_tmpb_tl = 1
5791       { \bool_set_false:N \g_tmpa_bool }
5792       {
5793         \seq_if_in:NxT
5794         \l_@@_corners_cells_seq
5795         { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
5796         { \bool_set_false:N \g_tmpa_bool }
5797       }
5798     }
5799   }
5800 }

```

```

5801 \cs_new_protected:Npn \@@_vline_ii:
5802 {
5803   \keys_set:nV { NiceMatrix / RulesBis } \l_@@_other_keys_tl
5804   \bool_if:NTF \l_@@_dotted_bool

```

```

5805 \@@_vline_iv:
5806 {
5807   \tl_if_empty:NTF \l_@@_tikz_rule_tl
5808     \@@_vline_iii:
5809     \@@_vline_v:
5810 }
5811 }

```

First the case of a standard rule: the user has not used the key `dotted` nor the key `tikz`.

```

5812 \cs_new_protected:Npn \@@_vline_iii:
5813 {
5814   \pgfpicture
5815   \pgfrememberpicturepositiononpagetrue
5816   \pgf@relevantforpicturesizefalse
5817   \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
5818   \dim_set_eq:NN \l_tmpa_dim \pgf@y
5819   \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
5820   \dim_set:Nn \l_tmpb_dim
5821   {
5822     \pgf@x
5823     - 0.5 \l_@@_rule_width_dim
5824     +
5825     ( \arrayrulewidth * \l_@@_multiplicity_int
5826       + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
5827   }
5828   \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
5829   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
5830   \bool_lazy_all:nT
5831   {
5832     { \int_compare_p:nNn \l_@@_multiplicity_int > 1 }
5833     { \cs_if_exist_p:N \CT@drsc@ }
5834     { ! \tl_if_blank_p:V \CT@drsc@ }
5835   }
5836   {
5837     \group_begin:
5838     \CT@drsc@
5839     \dim_add:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }
5840     \dim_sub:Nn \l_@@_tmpc_dim { 0.5 \arrayrulewidth }
5841     \dim_set:Nn \l_@@_tmpd_dim
5842     {
5843       \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
5844       * ( \l_@@_multiplicity_int - 1 )
5845     }
5846     \pgfpathrectanglecorners
5847     { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
5848     { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
5849     \pgfusepath { fill }
5850     \group_end:
5851   }
5852   \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
5853   \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
5854   \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
5855   {
5856     \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
5857     \dim_sub:Nn \l_tmpb_dim \doublerulesep
5858     \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
5859     \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
5860   }
5861   \CT@arc@
5862   \pgfsetlinewidth { 1.1 \arrayrulewidth }
5863   \pgfsetrectcap
5864   \pgfusepathqstroke
5865   \endpgfpicture

```

```
5866 }
```

The following code is for the case of a dotted rule (with our system of rounded dots).

```
5867 \cs_new_protected:Npn \@@_vline_iv:
5868 {
5869   \pgfpicture
5870   \pgfrememberpicturepositiononpagetrue
5871   \pgf@relevantforpicturesizefalse
5872   \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
5873   \dim_set:Nn \l_@@_x_initial_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
5874   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
5875   \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
5876   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
5877   \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
5878   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
5879   \CT@arc@
5880   \@@_draw_line:
5881   \endpgfpicture
5882 }
```

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`).

```
5883 \cs_new_protected:Npn \@@_vline_v:
5884 {
5885   \begin {tikzpicture }
5886   \pgfrememberpicturepositiononpagetrue
5887   \pgf@relevantforpicturesizefalse
5888   \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
5889   \dim_set_eq:NN \l_tmpa_dim \pgf@y
5890   \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
5891   \dim_set:Nn \l_tmpb_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
5892   \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
5893   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
5894   \exp_args:NV \tikzset \l_@@_tikz_rule_tl
5895   \use:x { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
5896     ( \l_tmpb_dim , \l_tmpa_dim ) --
5897     ( \l_tmpb_dim , \l_@@_tmpc_dim ) ;
5898   \end { tikzpicture }
5899 }
```

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).

```
5900 \cs_new_protected:Npn \@@_draw_vlines:
5901 {
5902   \int_step_inline:nnn
5903     {
5904       \bool_if:nTF { ! \g_@@_delims_bool && ! \l_@@_except_borders_bool }
5905         1 2
5906     }
5907     {
5908       \bool_if:nTF { ! \g_@@_delims_bool && ! \l_@@_except_borders_bool }
5909         { \int_eval:n { \c@jCol + 1 } }
5910         \c@jCol
5911     }
5912     {
5913       \tl_if_eq:NnF \l_@@_vlines_clist { all }
5914       { \clist_if_in:NnT \l_@@_vlines_clist { ##1 } }
5915       { \@@_vline:n { position = ##1 , total-width = \arrayrulewidth } }
5916     }
5917 }
```

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}`.

```
5918 \cs_new_protected:Npn \@@_hline:n #1
5919 {
```

The group is for the options.

```
5920   \group_begin:
5921   \int_zero_new:N \l_@@_end_int
5922   \int_set_eq:NN \l_@@_end_int \c_jCol
5923   \keys_set_known:nN { NiceMatrix / Rules } { #1 } \l_@@_other_keys_tl
5924   \@@_hline_i:
5925   \group_end:
5926 }
```

```
5927 \cs_new_protected:Npn \@@_hline_i:
5928 {
5929   \int_zero_new:N \l_@@_local_start_int
5930   \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`.

```
5931   \tl_set:Nx \l_tmpa_tl { \int_use:N \l_@@_position_int }
5932   \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
5933     \l_tmpb_tl
5934   {
```

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.

```
5935     \bool_gset_true:N \g_tmpa_bool
5936     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
5937       { \@@_test_hline_in_block:nnnnn ##1 }
5938     \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
5939       { \@@_test_hline_in_block:nnnnn ##1 }
5940     \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
5941       { \@@_test_hline_in_stroken_block:nnnn ##1 }
5942     \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_h:
5943     \bool_if:NTF \g_tmpa_bool
5944       {
5945         \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.

```
5946         { \int_set:Nn \l_@@_local_start_int \l_tmpb_tl }
5947       }
5948     {
5949       \int_compare:nNnT \l_@@_local_start_int > 0
5950       {
5951         \int_set:Nn \l_@@_local_end_int { \l_tmpb_tl - 1 }
5952         \@@_hline_ii:
5953         \int_zero:N \l_@@_local_start_int
5954       }
5955     }
5956   }
5957   \int_compare:nNnT \l_@@_local_start_int > 0
5958   {
5959     \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
5960     \@@_hline_ii:
5961   }
5962 }
```

```

5963 \cs_new_protected:Npn \@@_test_in_corner_h:
5964 {
5965   \int_compare:nNnTF \l_tmpa_tl = { \int_eval:n { \c@iRow + 1 } }
5966   {
5967     \seq_if_in:NxT
5968     \l_@@_corners_cells_seq
5969     { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
5970     { \bool_set_false:N \g_tmpa_bool }
5971   }
5972   {
5973     \seq_if_in:NxT
5974     \l_@@_corners_cells_seq
5975     { \l_tmpa_tl - \l_tmpb_tl }
5976     {
5977       \int_compare:nNnTF \l_tmpa_tl = 1
5978       { \bool_set_false:N \g_tmpa_bool }
5979       {
5980         \seq_if_in:NxT
5981         \l_@@_corners_cells_seq
5982         { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
5983         { \bool_set_false:N \g_tmpa_bool }
5984       }
5985     }
5986   }
5987 }

5988 \cs_new_protected:Npn \@@_hline_ii:
5989 {
5990   \keys_set:nV { NiceMatrix / RulesBis } \l_@@_other_keys_tl
5991   \bool_if:NTF \l_@@_dotted_bool
5992   \@@_hline_iv:
5993   {
5994     \tl_if_empty:NTF \l_@@_tikz_rule_tl
5995     \@@_hline_iii:
5996     \@@_hline_v:
5997   }
5998 }

```

First the case of a standard rule (without the keys dotted and tikz).

```

5999 \cs_new_protected:Npn \@@_hline_iii:
6000 {
6001   \pgfpicture
6002   \pgfrememberpicturepositiononpagetrue
6003   \pgf@relevantforpicturesizefalse
6004   \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6005   \dim_set_eq:NN \l_tmpa_dim \pgf@x
6006   \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6007   \dim_set:Nn \l_tmpb_dim
6008   {
6009     \pgf@y
6010     - 0.5 \l_@@_rule_width_dim
6011     +
6012     ( \arrayrulewidth * \l_@@_multiplicity_int
6013       + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6014   }
6015   \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6016   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
6017   \bool_lazy_all:nT
6018   {
6019     { \int_compare_p:nNn \l_@@_multiplicity_int > 1 }
6020     { \cs_if_exist_p:N \CT@drsc@ }
6021     { ! \tl_if_blank_p:V \CT@drsc@ }

```

```

6022     }
6023     {
6024         \group_begin:
6025         \CT@drsc@
6026         \dim_set:Nn \l_@@_tmpd_dim
6027         {
6028             \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
6029             * ( \l_@@_multiplicity_int - 1 )
6030         }
6031         \pgfpathrectanglecorners
6032         { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6033         { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
6034         \pgfusepathqfill
6035         \group_end:
6036     }
6037     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6038     \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
6039     \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6040     {
6041         \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
6042         \dim_sub:Nn \l_tmpb_dim \doublerulesep
6043         \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6044         \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
6045     }
6046     \CT@arc@
6047     \pgfsetlinewidth { 1.1 \arrayrulewidth }
6048     \pgfsetrectcap
6049     \pgfusepathqstroke
6050     \endpgfpicture
6051 }

```

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}
```

$$\begin{array}{|cccc|} \hline 1 & 2 & 3 & 4 \\ \hline 1 & 2 & 3 & 4 \\ \hline \cdots & \cdots & \cdots & \cdots \\ \hline 1 & 2 & 3 & 4 \\ \hline \end{array}$$

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}
```

$$\begin{array}{|cccc|} \hline 1 & 2 & 3 & 4 \\ \hline 1 & 2 & 3 & 4 \\ \hline \cdots & \cdots & \cdots & \cdots \\ \hline 1 & 2 & 3 & 4 \\ \hline \end{array}$$

```
6052 \cs_new_protected:Npn \l_@@_hline_iv:
```

```
6053 {
```

```
6054     \pgfpicture
```

```
6055     \pgfrememberpicturepositiononpagetrue
```

```
6056     \pgf@relevantforpicturesizefalse
```

```
6057     \l_@@_qpoint:n { row - \int_use:N \l_@@_position_int }
```

```
6058     \dim_set:Nn \l_@@_y_initial_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
```

```
6059     \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
```

```
6060     \l_@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
```

```
6061     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
```

```
6062     \int_compare:nNnT \l_@@_local_start_int = 1
```

```
6063     {
```

```
6064         \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
```

```

6065 \bool_if:NF \g_@@_delims_bool
6066 { \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 $0.5 \text{ } \l_@@_xdots_inter_dim$ is *ad hoc* for a better result.

```

6067 \tl_if_eq:NnF \g_@@_left_delim_tl (
6068 { \dim_add:Nn \l_@@_x_initial_dim { 0.5 \l_@@_xdots_inter_dim } }
6069 }
6070 \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6071 \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
6072 \int_compare:nNnT \l_@@_local_end_int = \c@jCol
6073 {
6074 \dim_add:Nn \l_@@_x_final_dim \l_@@_right_margin_dim
6075 \bool_if:NF \g_@@_delims_bool
6076 { \dim_add:Nn \l_@@_x_final_dim \arraycolsep }
6077 \tl_if_eq:NnF \g_@@_right_delim_tl )
6078 { \dim_gsub:Nn \l_@@_x_final_dim { 0.5 \l_@@_xdots_inter_dim } }
6079 }
6080 \CT@arc@
6081 \@@_draw_line:
6082 \endpgfpicture
6083 }

```

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`).

```

6084 \cs_new_protected:Npn \@@_hline_v:
6085 {
6086 \begin { tikzpicture }
6087 \pgfrememberpicturepositiononpagetrue
6088 \pgf@relevantforpicturesizefalse
6089 \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6090 \dim_set_eq:NN \l_tmpa_dim \pgf@x
6091 \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6092 \dim_set:Nn \l_tmpb_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
6093 \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6094 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
6095 \exp_args:NV \tikzset \l_@@_tikz_rule_tl
6096 \use:x { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6097 ( \l_tmpa_dim , \l_tmpb_dim ) --
6098 ( \l_@@_tmpc_dim , \l_tmpb_dim ) ;
6099 \end { tikzpicture }
6100 }

```

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).

```

6101 \cs_new_protected:Npn \@@_draw_hlines:
6102 {
6103 \int_step_inline:nnn
6104 {
6105 \bool_if:NTF { ! \g_@@_delims_bool && ! \l_@@_except_borders_bool }
6106 1 2
6107 }
6108 {
6109 \bool_if:NTF { ! \g_@@_delims_bool && ! \l_@@_except_borders_bool }
6110 { \int_eval:n { \c@iRow + 1 } }
6111 \c@iRow
6112 }
6113 {
6114 \tl_if_eq:NnF \l_@@_hlines_clist { all }
6115 { \clist_if_in:NnT \l_@@_hlines_clist { ##1 } }
6116 { \@@_hline:n { position = ##1 , total-width = \arrayrulewidth } }

```

```

6117     }
6118 }

```

The command `\@@_Hline:` will be linked to `\Hline` in the environments of `nicematrix`.

```

6119 \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.

```

6120 \cs_set:Npn \@@_Hline_i:n #1
6121 {
6122   \peek_remove_spaces:n
6123   {
6124     \peek_meaning:NTF \Hline
6125     { \@@_Hline_ii:nn { #1 + 1 } }
6126     { \@@_Hline_iii:n { #1 } }
6127   }
6128 }
6129 \cs_set:Npn \@@_Hline_ii:nn #1 #2 { \@@_Hline_i:n { #1 } }
6130 \cs_set:Npn \@@_Hline_iii:n #1
6131 { \@@_collect_options:n { \@@_Hline_iv:nn { #1 } } }
6132 \cs_set:Npn \@@_Hline_iv:nn #1 #2
6133 {
6134   \@@_compute_rule_width:n { multiplicity = #1 , #2 }
6135   \skip_vertical:n { \l_@@_rule_width_dim }
6136   \tl_gput_right:Nx \g_@@_pre_code_after_tl
6137   {
6138     \@@_hline:n
6139     {
6140       multiplicity = #1 ,
6141       position = \int_eval:n { \c@iRow + 1 } ,
6142       total-width = \dim_use:N \l_@@_rule_width_dim ,
6143       #2
6144     }
6145   }
6146   \egroup
6147 }

```

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).

```

6148 \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`).

```

6149 \cs_new_protected:Npn \@@_custom_line:n #1
6150 {
6151   \str_clear_new:N \l_@@_command_str
6152   \str_clear_new:N \l_@@_ccommand_str
6153   \str_clear_new:N \l_@@_letter_str
6154   \tl_clear_new:N \l_@@_other_keys_tl
6155   \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.

```

6156 \bool_lazy_all:nTF
6157 {
6158   { \str_if_empty_p:N \l_@@_letter_str }
6159   { \str_if_empty_p:N \l_@@_command_str }
6160   { \str_if_empty_p:N \l_@@_ccommand_str }
6161 }
6162 { \@@_error:n { No~letter~and~no~command } }
6163 { \exp_args:NV \@@_custom_line_i:n \l_@@_other_keys_tl }
6164 }
6165 \keys_define:nn { NiceMatrix / custom-line }
6166 {
6167   letter .str_set:N = \l_@@_letter_str ,
6168   letter .value_required:n = true ,
6169   command .str_set:N = \l_@@_command_str ,
6170   command .value_required:n = true ,
6171   ccommand .str_set:N = \l_@@_ccommand_str ,
6172   ccommand .value_required:n = true ,
6173 }
6174 \cs_new_protected:Npn \@@_custom_line_i:n #1
6175 {

```

The following flags will be raised when the keys `tikz`, `dotted` and `color` are used (in the `custom-line`).

```

6176 \bool_set_false:N \l_@@_tikz_rule_bool
6177 \bool_set_false:N \l_@@_dotted_rule_bool
6178 \bool_set_false:N \l_@@_color_bool
6179 \keys_set:nn { NiceMatrix / custom-line-bis } { #1 }
6180 \bool_if:NT \l_@@_tikz_rule_bool
6181 {
6182   \IfPackageLoadedTF { tikz }
6183   { }
6184   { \@@_error:n { tikz~in~custom~line~without~tikz } }
6185   \bool_if:NT \l_@@_color_bool
6186   { \@@_error:n { color~in~custom~line~with~tikz } }
6187 }
6188 \bool_if:nT
6189 {
6190   \int_compare_p:nNn \l_@@_multiplicity_int > 1
6191   && \l_@@_dotted_rule_bool
6192 }
6193 { \@@_error:n { key~multiplicity~with~dotted } }
6194 \str_if_empty:NF \l_@@_letter_str
6195 {
6196   \int_compare:nTF { \str_count:N \l_@@_letter_str != 1 }
6197   { \@@_error:n { Several~letters } }
6198   {
6199     \exp_args:NnV \tl_if_in:NnTF
6200     \c_@@_forbidden_letters_str \l_@@_letter_str
6201     { \@@_error:n { Forbidden~letter } }
6202     {

```

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.

```

6203 \keys_define:nx { NiceMatrix / ColumnTypes }
6204 {
6205   \l_@@_letter_str .code:n =

```

```

6206             { \l_@@_v_custom_line:n { \exp_not:n { #1 } } }
6207         }
6208     }
6209 }
6210 }
6211 \str_if_empty:NF \l_@@_command_str { \l_@@_h_custom_line:n { #1 } }
6212 \str_if_empty:NF \l_@@_ccommand_str { \l_@@_c_custom_line:n { #1 } }
6213 }
6214 \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.

```

6215 \keys_define:nn { NiceMatrix / custom-line-bis }
6216 {
6217     multiplicity .int_set:N = \l_@@_multiplicity_int ,
6218     multiplicity .initial:n = 1 ,
6219     multiplicity .value_required:n = true ,
6220     color .code:n = \bool_set_true:N \l_@@_color_bool ,
6221     color .value_required:n = true ,
6222     tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6223     tikz .value_required:n = true ,
6224     dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6225     dotted .value_forbidden:n = true ,
6226     total-width .code:n = { } ,
6227     total-width .value_required:n = true ,
6228     width .code:n = { } ,
6229     width .value_required:n = true ,
6230     sep-color .code:n = { } ,
6231     sep-color .value_required:n = true ,
6232     unknown .code:n = \@@_error:n { Unknown~key~for~custom-line }
6233 }

```

The following keys will indicate whether the keys `dotted`, `tikz` and `color` are used in the use of a `custom-line`.

```

6234 \bool_new:N \l_@@_dotted_rule_bool
6235 \bool_new:N \l_@@_tikz_rule_bool
6236 \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`.

```

6237 \keys_define:nn { NiceMatrix / custom-line-width }
6238 {
6239     multiplicity .int_set:N = \l_@@_multiplicity_int ,
6240     multiplicity .initial:n = 1 ,
6241     multiplicity .value_required:n = true ,
6242     tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6243     total-width .code:n = \dim_set:Nn \l_@@_rule_width_dim { #1 }
6244         \bool_set_true:N \l_@@_total_width_bool ,
6245     total-width .value_required:n = true ,
6246     width .meta:n = { total-width = #1 } ,
6247     dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6248 }

```

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`).

```

6249 \cs_new_protected:Npn \l_@@_h_custom_line:n #1
6250 {

```

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`.

```

6251 \cs_set:cpn { nicematrix - \l_@@_command_str }
6252 {
6253   \noalign
6254   {
6255     \@@_compute_rule_width:n { #1 }
6256     \skip_vertical:n { \l_@@_rule_width_dim }
6257     \tl_gput_right:Nx \g_@@_pre_code_after_tl
6258     {
6259       \@@_hline:n
6260       {
6261         #1 ,
6262         position = \int_eval:n { \c@iRow + 1 } ,
6263         total-width = \dim_use:N \l_@@_rule_width_dim
6264       }
6265     }
6266   }
6267 }
6268 \seq_put_left:NV \l_@@_custom_line_commands_seq \l_@@_command_str
6269 }

```

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`).

```

6270 \cs_new_protected:Npn \@@_c_custom_line:n #1
6271 {

```

Here, we need an expandable command since it begins with an `\noalign`.

```

6272 \exp_args:Nc \NewExpandableDocumentCommand
6273 { nicematrix - \l_@@_ccommand_str }
6274 { 0 { } m }
6275 {
6276   \noalign
6277   {
6278     \@@_compute_rule_width:n { #1 , ##1 }
6279     \skip_vertical:n { \l_@@_rule_width_dim }
6280     \clist_map_inline:nn
6281     { ##2 }
6282     { \@@_c_custom_line_i:nn { #1 , ##1 } { ####1 } }
6283   }
6284 }
6285 \seq_put_left:NV \l_@@_custom_line_commands_seq \l_@@_ccommand_str
6286 }

```

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*.

```

6287 \cs_new_protected:Npn \@@_c_custom_line_i:nn #1 #2
6288 {
6289   \str_if_in:nnTF { #2 } { - }
6290   { \@@_cut_on_hyphen:w #2 \q_stop }
6291   { \@@_cut_on_hyphen:w #2 - #2 \q_stop }
6292   \tl_gput_right:Nx \g_@@_pre_code_after_tl
6293   {
6294     \@@_hline:n
6295     {
6296       #1 ,
6297       start = \l_tmpa_tl ,
6298       end = \l_tmpb_tl ,
6299       position = \int_eval:n { \c@iRow + 1 } ,
6300       total-width = \dim_use:N \l_@@_rule_width_dim
6301     }

```

```

6302     }
6303 }
6304 \cs_new_protected:Npn \@@_compute_rule_width:n #1
6305 {
6306     \bool_set_false:N \l_@@_tikz_rule_bool
6307     \bool_set_false:N \l_@@_total_width_bool
6308     \bool_set_false:N \l_@@_dotted_rule_bool
6309     \keys_set_known:nn { NiceMatrix / custom-line-width } { #1 }
6310     \bool_if:NF \l_@@_total_width_bool
6311     {
6312         \bool_if:NTF \l_@@_dotted_rule_bool
6313         { \dim_set:Nn \l_@@_rule_width_dim { 2 \l_@@_xdots_radius_dim } }
6314         {
6315             \bool_if:NF \l_@@_tikz_rule_bool
6316             {
6317                 \dim_set:Nn \l_@@_rule_width_dim
6318                 {
6319                     \arrayrulewidth * \l_@@_multiplicity_int
6320                     + \doublerulesep * ( \l_@@_multiplicity_int - 1 )
6321                 }
6322             }
6323         }
6324     }
6325 }
6326 \cs_new_protected:Npn \@@_v_custom_line:n #1
6327 {
6328     \@@_compute_rule_width:n { #1 }

```

In the following line, the `\dim_use:N` is mandatory since we do an expansion.

```

6329     \tl_gput_right:Nx \g_@@_preamble_tl
6330     { \exp_not:N ! { \skip_horizontal:n { \dim_use:N \l_@@_rule_width_dim } } }
6331     \tl_gput_right:Nx \g_@@_pre_code_after_tl
6332     {
6333         \@@_vline:n
6334         {
6335             #1 ,
6336             position = \int_eval:n { \c@jCol + 1 } ,
6337             total-width = \dim_use:N \l_@@_rule_width_dim
6338         }
6339     }
6340 }
6341 \@@_custom_line:n
6342 { 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.

```

6343 \cs_new_protected:Npn \@@_test_hline_in_block:nnnn #1 #2 #3 #4 #5
6344 {
6345     \bool_lazy_all:nT
6346     {
6347         { \int_compare_p:nNn \l_tmpa_tl > { #1 } }
6348         { \int_compare_p:nNn \l_tmpa_tl < { #3 + 1 } }
6349         { \int_compare_p:nNn \l_tmpb_tl > { #2 - 1 } }
6350         { \int_compare_p:nNn \l_tmpb_tl < { #4 + 1 } }
6351     }
6352     { \bool_gset_false:N \g_tmpa_bool }
6353 }

```

The same for vertical rules.

```

6354 \cs_new_protected:Npn \@@_test_vline_in_block:nnnnn #1 #2 #3 #4 #5
6355 {
6356   \bool_lazy_all:nT
6357   {
6358     { \int_compare_p:nNn \l_tmpa_tl > { #1 - 1 } }
6359     { \int_compare_p:nNn \l_tmpa_tl < { #3 + 1 } }
6360     { \int_compare_p:nNn \l_tmpb_tl > { #2 } }
6361     { \int_compare_p:nNn \l_tmpb_tl < { #4 + 1 } }
6362   }
6363   { \bool_gset_false:N \g_tmpa_bool }
6364 }

6365 \cs_new_protected:Npn \@@_test_hline_in_stroken_block:nnnn #1 #2 #3 #4
6366 {
6367   \bool_lazy_all:nT
6368   {
6369     {
6370       ( \int_compare_p:nNn \l_tmpa_tl = { #1 } )
6371       || ( \int_compare_p:nNn \l_tmpa_tl = { #3 + 1 } )
6372     }
6373     { \int_compare_p:nNn \l_tmpb_tl > { #2 - 1 } }
6374     { \int_compare_p:nNn \l_tmpb_tl < { #4 + 1 } }
6375   }
6376   { \bool_gset_false:N \g_tmpa_bool }
6377 }

6378 \cs_new_protected:Npn \@@_test_vline_in_stroken_block:nnnn #1 #2 #3 #4
6379 {
6380   \bool_lazy_all:nT
6381   {
6382     { \int_compare_p:nNn \l_tmpa_tl > { #1 - 1 } }
6383     { \int_compare_p:nNn \l_tmpa_tl < { #3 + 1 } }
6384     {
6385       ( \int_compare_p:nNn \l_tmpb_tl = { #2 } )
6386       || ( \int_compare_p:nNn \l_tmpb_tl = { #4 + 1 } )
6387     }
6388   }
6389   { \bool_gset_false:N \g_tmpa_bool }
6390 }

```

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.

```

6391 \cs_new_protected:Npn \@@_compute_corners:
6392 {

```

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.

```

6393   \seq_clear_new:N \l_@@_corners_cells_seq
6394   \clist_map_inline:Nn \l_@@_corners_clist
6395   {
6396     \str_case:nnF { ##1 }
6397     {
6398       { NW }
6399       { \@@_compute_a_corner:nnnnnn 1 1 1 1 \c@iRow \c@jCol }
6400       { NE }
6401       { \@@_compute_a_corner:nnnnnn 1 \c@jCol 1 { -1 } \c@iRow 1 }
6402       { SW }

```

```

6403         { \@@_compute_a_corner:nnnnnn \c@iRow 1 { -1 } 1 1 \c@jCol }
6404         { SE }
6405         { \@@_compute_a_corner:nnnnnn \c@iRow \c@jCol { -1 } { -1 } 1 1 }
6406     }
6407     { \@@_error:nn { bad~corner } { ##1 } }
6408 }

```

Even if the user has used the key `corners` the list of cells in the corners may be empty.

```

6409     \seq_if_empty:NF \l_@@_corners_cells_seq
6410     {

```

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.

```

6411         \tl_gput_right:Nx \g_@@_aux_tl
6412         {
6413             \seq_set_from_clist:Nn \exp_not:N \l_@@_corners_cells_seq
6414             { \seq_use:Nnnn \l_@@_corners_cells_seq , , , }
6415         }
6416     }
6417 }

```

“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.

```

6418 \cs_new_protected:Npn \@@_compute_a_corner:nnnnnn #1 #2 #3 #4 #5 #6
6419 {

```

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.

```

6420     \bool_set_false:N \l_tmpa_bool
6421     \int_zero_new:N \l_@@_last_empty_row_int
6422     \int_set:Nn \l_@@_last_empty_row_int { #1 }
6423     \int_step_inline:nnnn { #1 } { #3 } { #5 }
6424     {
6425         \@@_test_if_cell_in_a_block:nn { ##1 } { \int_eval:n { #2 } }
6426         \bool_lazy_or:nnTF
6427         {
6428             \cs_if_exist_p:c
6429             { pgf @ sh @ ns @ \@@_env: - ##1 - \int_eval:n { #2 } }
6430         }
6431         \l_tmpb_bool
6432         { \bool_set_true:N \l_tmpa_bool }
6433         {
6434             \bool_if:NF \l_tmpa_bool
6435             { \int_set:Nn \l_@@_last_empty_row_int { ##1 } }
6436         }
6437     }

```

Now, you determine the last empty cell in the row of number 1.

```

6438 \bool_set_false:N \l_tmpa_bool
6439 \int_zero_new:N \l_@@_last_empty_column_int
6440 \int_set:Nn \l_@@_last_empty_column_int { #2 }
6441 \int_step_inline:nnnn { #2 } { #4 } { #6 }
6442 {
6443   \@@_test_if_cell_in_a_block:nn { \int_eval:n { #1 } } { ##1 }
6444   \bool_lazy_or:nnTF
6445     \l_tmpb_bool
6446     {
6447       \cs_if_exist_p:c
6448         { pgf @ sh @ ns @ \@@_env: - \int_eval:n { #1 } - ##1 }
6449     }
6450     { \bool_set_true:N \l_tmpa_bool }
6451     {
6452       \bool_if:NF \l_tmpa_bool
6453       { \int_set:Nn \l_@@_last_empty_column_int { ##1 } }
6454     }
6455 }

```

Now, we loop over the rows.

```

6456 \int_step_inline:nnnn { #1 } { #3 } \l_@@_last_empty_row_int
6457 {

```

We treat the row number ##1 with another loop.

```

6458   \bool_set_false:N \l_tmpa_bool
6459   \int_step_inline:nnnn { #2 } { #4 } \l_@@_last_empty_column_int
6460   {
6461     \@@_test_if_cell_in_a_block:nn { ##1 } { #####1 }
6462     \bool_lazy_or:nnTF
6463       \l_tmpb_bool
6464       {
6465         \cs_if_exist_p:c
6466           { pgf @ sh @ ns @ \@@_env: - ##1 - #####1 }
6467       }
6468       { \bool_set_true:N \l_tmpa_bool }
6469       {
6470         \bool_if:NF \l_tmpa_bool
6471         {
6472           \int_set:Nn \l_@@_last_empty_column_int { #####1 }
6473           \seq_put_right:Nn
6474             \l_@@_corners_cells_seq
6475             { ##1 - #####1 }
6476         }
6477       }
6478     }
6479   }
6480 }

```

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`).

```

6481 \cs_new_protected:Npn \@@_test_if_cell_in_a_block:nn #1 #2
6482 {
6483   \int_set:Nn \l_tmpa_int { #1 }
6484   \int_set:Nn \l_tmpb_int { #2 }
6485   \bool_set_false:N \l_tmpb_bool
6486   \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6487     { \@@_test_if_cell_in_block:nnnnnn \l_tmpa_int \l_tmpb_int ##1 }
6488 }
6489 \cs_new_protected:Npn \@@_test_if_cell_in_block:nnnnnn #1 #2 #3 #4 #5 #6 #7
6490 {
6491   \int_compare:nNnT { #3 } < { \int_eval:n { #1 + 1 } }

```

```

6492 {
6493   \int_compare:nNnT { #1 } < { \int_eval:n { #5 + 1 } }
6494   {
6495     \int_compare:nNnT { #4 } < { \int_eval:n { #2 + 1 } }
6496     {
6497       \int_compare:nNnT { #2 } < { \int_eval:n { #6 + 1 } }
6498       { \bool_set_true:N \l_tmpb_bool }
6499     }
6500   }
6501 }
6502 }

```

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.

```

6503 \bool_new:N \l_@@_block_auto_columns_width_bool

```

Up to now, there is only one option available for the environment {NiceMatrixBlock}.

```

6504 \keys_define:nn { NiceMatrix / NiceMatrixBlock }
6505 {
6506   auto-columns-width .code:n =
6507   {
6508     \bool_set_true:N \l_@@_block_auto_columns_width_bool
6509     \dim_gzero_new:N \g_@@_max_cell_width_dim
6510     \bool_set_true:N \l_@@_auto_columns_width_bool
6511   }
6512 }

6513 \NewDocumentEnvironment { NiceMatrixBlock } { ! 0 { } }
6514 {
6515   \int_gincr:N \g_@@_NiceMatrixBlock_int
6516   \dim_zero:N \l_@@_columns_width_dim
6517   \keys_set:nn { NiceMatrix / NiceMatrixBlock } { #1 }
6518   \bool_if:NT \l_@@_block_auto_columns_width_bool
6519   {
6520     \cs_if_exist:cT
6521     { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
6522     {
6523       \exp_args:NNx \dim_set:Nn \l_@@_columns_width_dim
6524       {
6525         \use:c
6526         { @@_max_cell_width _ \int_use:N \g_@@_NiceMatrixBlock_int }
6527       }
6528     }
6529   }
6530 }

```

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).

```

6531 {
6532   \legacy_if:nTF { measuring@ }

```

If `{NiceMatrixBlock}` is used in an environment of `amsmath` such as `{align}`: cf. question 694957 on TeX StackExchange. The most important line in that case is the following one.

```

6533     { \int_gdecr:N \g_@@_NiceMatrixBlock_int }
6534     {
6535         \bool_if:NT \l_@@_block_auto_columns_width_bool
6536         {
6537             \iow_shipout:Nn \@mainaux \ExplSyntaxOn
6538             \iow_shipout:Nx \@mainaux
6539             {
6540                 \cs_gset:cpn
6541                 { @@ _ 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.

```

6542         { \dim_eval:n { \g_@@_max_cell_width_dim + \arrayrulewidth } }
6543     }
6544     \iow_shipout:Nn \@mainaux \ExplSyntaxOff
6545 }
6546 }
6547 \ignorespacesafterend
6548 }

```

26 The extra nodes

First, two variants of the functions `\dim_min:nn` and `\dim_max:nn`.

```

6549 \cs_generate_variant:Nn \dim_min:nn { v n }
6550 \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).

```

6551 \cs_new_protected:Npn \@@_create_extra_nodes:
6552 {
6553     \bool_if:nTF \l_@@_medium_nodes_bool
6554     {
6555         \bool_if:NTF \l_@@_large_nodes_bool
6556         \@@_create_medium_and_large_nodes:
6557         \@@_create_medium_nodes:
6558     }
6559     { \bool_if:NT \l_@@_large_nodes_bool \@@_create_large_nodes: }
6560 }

```

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`.

```

6561 \cs_new_protected:Npn \@@_computations_for_medium_nodes:
6562 {
6563   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6564   {
6565     \dim_zero_new:c { \l_@@_row_\@@_i: _min_dim }
6566     \dim_set_eq:cN { \l_@@_row_\@@_i: _min_dim } \c_max_dim
6567     \dim_zero_new:c { \l_@@_row_\@@_i: _max_dim }
6568     \dim_set:cn { \l_@@_row_\@@_i: _max_dim } { - \c_max_dim }
6569   }
6570   \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
6571   {
6572     \dim_zero_new:c { \l_@@_column_\@@_j: _min_dim }
6573     \dim_set_eq:cN { \l_@@_column_\@@_j: _min_dim } \c_max_dim
6574     \dim_zero_new:c { \l_@@_column_\@@_j: _max_dim }
6575     \dim_set:cn { \l_@@_column_\@@_j: _max_dim } { - \c_max_dim }
6576   }

```

We begin the two nested loops over the rows and the columns of the array.

```

6577   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6578   {
6579     \int_step_variable:nnNn
6580     \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.

```

6581     {
6582       \cs_if_exist:cT
6583       { \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`.

```

6584     {
6585       \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { south-west }
6586       \dim_set:cn { \l_@@_row_\@@_i: _min_dim }
6587       { \dim_min:vn { \l_@@_row _ \@@_i: _min_dim } \pgf@y }
6588       \seq_if_in:NxF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
6589       {
6590         \dim_set:cn { \l_@@_column _ \@@_j: _min_dim }
6591         { \dim_min:vn { \l_@@_column _ \@@_j: _min_dim } \pgf@x }
6592       }

```

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`.

```

6593       \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { north-east }
6594       \dim_set:cn { \l_@@_row _ \@@_i: _ max_dim }
6595       { \dim_max:vn { \l_@@_row _ \@@_i: _ max_dim } \pgf@y }
6596       \seq_if_in:NxF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
6597       {
6598         \dim_set:cn { \l_@@_column _ \@@_j: _ max_dim }
6599         { \dim_max:vn { \l_@@_column _ \@@_j: _ max_dim } \pgf@x }
6600       }
6601     }
6602   }
6603 }

```

Now, we have to deal with empty rows or empty columns since we don't have created nodes in such rows and columns.

```

6604   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6605   {
6606     \dim_compare:nNnT
6607     { \dim_use:c { \l_@@_row _ \@@_i: _ min _ dim } } = \c_max_dim
6608     {
6609       \@@_qpoint:n { row - \@@_i: - base }

```

```

6610         \dim_set:cn { l_@@_row _ \@@_i: _ max _ dim } \pgf@y
6611         \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim } \pgf@y
6612     }
6613 }
6614 \int_step_variable:nNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
6615 {
6616     \dim_compare:nNnT
6617     { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } } = \c_max_dim
6618     {
6619         \@@_qpoint:n { col - \@@_j: }
6620         \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim } \pgf@y
6621         \dim_set:cn { l_@@_column _ \@@_j: _ min _ dim } \pgf@y
6622     }
6623 }
6624 }

```

Here is the command `\@@_create_medium_nodes:`. When this command is used, the “medium nodes” are created.

```

6625 \cs_new_protected:Npn \@@_create_medium_nodes:
6626 {
6627     \pgfpicture
6628     \pgfrememberpicturepositiononpagetrue
6629     \pgf@relevantforpicturesizefalse
6630     \@@_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”.

```

6631     \tl_set:Nn \l_@@_suffix_tl { -medium }
6632     \@@_create_nodes:
6633     \endpgfpicture
6634 }

```

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:`.

```

6635 \cs_new_protected:Npn \@@_create_large_nodes:
6636 {
6637     \pgfpicture
6638     \pgfrememberpicturepositiononpagetrue
6639     \pgf@relevantforpicturesizefalse
6640     \@@_computations_for_medium_nodes:
6641     \@@_computations_for_large_nodes:
6642     \tl_set:Nn \l_@@_suffix_tl { - large }
6643     \@@_create_nodes:
6644     \endpgfpicture
6645 }
6646 \cs_new_protected:Npn \@@_create_medium_and_large_nodes:
6647 {
6648     \pgfpicture
6649     \pgfrememberpicturepositiononpagetrue
6650     \pgf@relevantforpicturesizefalse
6651     \@@_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”.

```

6652     \tl_set:Nn \l_@@_suffix_tl { - medium }
6653     \@@_create_nodes:

```

¹⁴If we want to create both, we have to use `\@@_create_medium_and_large_nodes:`

```

6654     \@@_computations_for_large_nodes:
6655     \tl_set:Nn \l_@@_suffix_tl { - large }
6656     \@@_create_nodes:
6657     \endpgfpicture
6658 }

```

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.

```

6659 \cs_new_protected:Npn \@@_computations_for_large_nodes:
6660 {
6661     \int_set:Nn \l_@@_first_row_int 1
6662     \int_set:Nn \l_@@_first_col_int 1

```

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`.

```

6663     \int_step_variable:nNn { \c@iRow - 1 } \@@_i:
6664     {
6665         \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim }
6666         {
6667             (
6668                 \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } +
6669                 \dim_use:c { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
6670             )
6671             / 2
6672         }
6673         \dim_set_eq:cc { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
6674         { l_@@_row _ \@@_i: _ min_dim }
6675     }
6676     \int_step_variable:nNn { \c@jCol - 1 } \@@_j:
6677     {
6678         \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim }
6679         {
6680             (
6681                 \dim_use:c { l_@@_column _ \@@_j: _ max _ dim } +
6682                 \dim_use:c
6683                 { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
6684             )
6685             / 2
6686         }
6687         \dim_set_eq:cc { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
6688         { l_@@_column _ \@@_j: _ max _ dim }
6689     }

```

Here, we have to use `\dim_sub:cn` because of the number 1 in the name.

```

6690     \dim_sub:cn
6691     { l_@@_column _ 1 _ min _ dim }
6692     \l_@@_left_margin_dim
6693     \dim_add:cn
6694     { l_@@_column _ \int_use:N \c@jCol _ max _ dim }
6695     \l_@@_right_margin_dim
6696 }

```

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).

```

6697 \cs_new_protected:Npn \@@_create_nodes:
6698 {
6699     \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6700     {

```

```

6701 \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
6702 {

```

We draw the rectangular node for the cell ($\backslash\@@_i-\backslash\@@_j$).

```

6703 \@@_pgf_rect_node:nnnnn
6704 { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
6705 { \dim_use:c { l_@@_column_ \@@_j: _min_dim } }
6706 { \dim_use:c { l_@@_row_ \@@_i: _min_dim } }
6707 { \dim_use:c { l_@@_column_ \@@_j: _max_dim } }
6708 { \dim_use:c { l_@@_row_ \@@_i: _max_dim } }
6709 \str_if_empty:NF \l_@@_name_str
6710 {
6711 \pgfnodealias
6712 { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
6713 { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
6714 }
6715 }
6716 }

```

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 .

```

6717 \cs_if_exist_use:NF
6718 \seq_map_pairwise_function:NNN
6719 \seq_mapthread_function:NNN
6720 \g_@@_multicolumn_cells_seq
6721 \g_@@_multicolumn_sizes_seq
6722 \@@_node_for_multicolumn:nn
6723 }

```

```

6724 \cs_new_protected:Npn \@@_extract_coords_values: #1 - #2 \q_stop
6725 {
6726 \cs_set_nopar:Npn \@@_i: { #1 }
6727 \cs_set_nopar:Npn \@@_j: { #2 }
6728 }

```

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”).

```

6729 \cs_new_protected:Npn \@@_node_for_multicolumn:nn #1 #2
6730 {
6731 \@@_extract_coords_values: #1 \q_stop
6732 \@@_pgf_rect_node:nnnnn
6733 { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
6734 { \dim_use:c { l_@@_column_ \@@_j: _min_dim } }
6735 { \dim_use:c { l_@@_row_ \@@_i: _min_dim } }
6736 { \dim_use:c { l_@@_column_ \int_eval:n { \@@_j: +#2-1 } _max_dim } }
6737 { \dim_use:c { l_@@_row_ \@@_i: _max_dim } }
6738 \str_if_empty:NF \l_@@_name_str
6739 {
6740 \pgfnodealias
6741 { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
6742 { \int_use:N \g_@@_env_int - \@@_i: - \@@_j: \l_@@_suffix_tl }
6743 }
6744 }

```

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.

```

6745 \keys_define:nn { NiceMatrix / Block / FirstPass }
6746 {
6747   l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
6748   l .value_forbidden:n = true ,
6749   r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
6750   r .value_forbidden:n = true ,
6751   c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
6752   c .value_forbidden:n = true ,
6753   L .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
6754   L .value_forbidden:n = true ,
6755   R .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
6756   R .value_forbidden:n = true ,
6757   C .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
6758   C .value_forbidden:n = true ,
6759   t .code:n = \str_set:Nn \l_@@_vpos_of_block_str t ,
6760   t .value_forbidden:n = true ,
6761   T .code:n = \str_set:Nn \l_@@_vpos_of_block_str T ,
6762   T .value_forbidden:n = true ,
6763   b .code:n = \str_set:Nn \l_@@_vpos_of_block_str b ,
6764   b .value_forbidden:n = true ,
6765   B .code:n = \str_set:Nn \l_@@_vpos_of_block_str B ,
6766   B .value_forbidden:n = true ,
6767   color .code:n =
6768     \@@_color:n { #1 }
6769     \tl_set_rescan:Nnn
6770       \l_@@_draw_tl
6771       { \char_set_catcode_other:N ! }
6772       { #1 } ,
6773   color .value_required:n = true ,
6774   respect-arraystretch .bool_set:N = \l_@@_respect_arraystretch_bool ,
6775   respect-arraystretch .default:n = true
6776 }

```

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.

```

6777 \cs_new_protected:Npn \@@_Block: { \@@_collect_options:n { \@@_Block_i: } }

6778 \NewExpandableDocumentCommand \@@_Block_i: { m m D < > { } +m }
6779 {

```

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).

```

6780   \peek_remove_spaces:n
6781   {
6782     \tl_if_blank:nTF { #2 }
6783     { \@@_Block_i 1-1 \q_stop }
6784     {
6785       \int_compare:nNnTF { \char_value_catcode:n { 45 } } = { 13 }
6786       \@@_Block_i_czech \@@_Block_i
6787       #2 \q_stop
6788     }
6789     { #1 } { #3 } { #4 }
6790   }
6791 }

```

With the following construction, we extract the values of i and j in the first mandatory argument of the command.

```

6792 \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`.

```

6793 {
6794   \char_set_catcode_active:N -
6795   \cs_new:Npn \@@_Block_i_czech #1-#2 \q_stop { \@@_Block_ii:nnnnn { #1 } { #2 } }
6796 }

```

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.

```

6797 \cs_new_protected:Npn \@@_Block_ii:nnnnn #1 #2 #3 #4 #5
6798 {

```

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).

```

6799   \bool_lazy_or:nnTF
6800     { \tl_if_blank_p:n { #1 } }
6801     { \str_if_eq_p:nn { #1 } { * } }
6802     { \int_set:Nn \l_tmpa_int { 100 } }
6803     { \int_set:Nn \l_tmpa_int { #1 } }
6804   \bool_lazy_or:nnTF
6805     { \tl_if_blank_p:n { #2 } }
6806     { \str_if_eq_p:nn { #2 } { * } }
6807     { \int_set:Nn \l_tmpb_int { 100 } }
6808     { \int_set:Nn \l_tmpb_int { #2 } }

```

If the block is mono-column.

```

6809   \int_compare:nNnTF \l_tmpb_int = 1
6810   {
6811     \str_if_empty:NTF \l_@@_hpos_cell_str
6812     { \str_set:Nn \l_@@_hpos_block_str c }
6813     { \str_set_eq:NN \l_@@_hpos_block_str \l_@@_hpos_cell_str }
6814   }
6815   { \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.

```

6816   \keys_set_known:nn { NiceMatrix / Block / FirstPass } { #3 }
6817   \tl_set:Nx \l_tmpa_tl
6818   {
6819     { \int_use:N \c@iRow }
6820     { \int_use:N \c@jCol }
6821     { \int_eval:n { \c@iRow + \l_tmpa_int - 1 } }
6822     { \int_eval:n { \c@jCol + \l_tmpb_int - 1 } }
6823   }

```

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).

```

6824   \bool_if:nTF
6825   {
6826     (
6827       \int_compare_p:nNn { \l_tmpa_int } = 1
6828       ||

```

```

6829         \int_compare:nNn { \l_tmpb_int } = 1
6830     )
6831     && ! \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.

```

6832         && ! \l_@@_X_column_bool
6833     }
6834     { \exp_args:Nxx \@@_Block_iv:nnnnn }
6835     { \exp_args:Nxx \@@_Block_v:nnnnn }
6836     { \l_tmpa_int } { \l_tmpb_int } { #3 } { #4 } { #5 }
6837 }

```

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.

```

6838 \cs_new_protected:Npn \@@_Block_iv:nnnnn #1 #2 #3 #4 #5
6839 {
6840     \int_gincr:N \g_@@_block_box_int
6841     \cs_set_protected_nopar:Npn \diagbox ##1 ##2
6842     {
6843         \tl_gput_right:Nx \g_@@_pre_code_after_tl
6844         {
6845             \@@_actually_diagbox:nnnnnn
6846             { \int_use:N \c@iRow }
6847             { \int_use:N \c@jCol }
6848             { \int_eval:n { \c@iRow + #1 - 1 } }
6849             { \int_eval:n { \c@jCol + #2 - 1 } }
6850             { \exp_not:n { ##1 } } { \exp_not:n { ##2 } }
6851         }
6852     }
6853     \box_gclear_new:c
6854     { 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!

```

6855     \hbox_gset:cn
6856     { g_@@_block_box_int \int_use:N \g_@@_block_box_int _ box }
6857 {

```

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}).

```

6858     \tl_if_empty:NTF \l_@@_color_tl
6859     { \int_compare:nNnT { #2 } = 1 \set@color }
6860     { \@@_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.

```

6861     \int_compare:nNnT { #1 } = 1

```

```

6862     {
6863         \int_compare:nNnTF \c@iRow = 0
6864         \l_@@_code_for_first_row_tl
6865         {
6866             \int_compare:nNnT \c@iRow = \l_@@_last_row_int
6867             \l_@@_code_for_last_row_tl
6868         }
6869         \g_@@_row_style_tl
6870     }
6871     \bool_if:NF \l_@@_respect_arraystretch_bool
6872     { \cs_set:Npn \arraystretch { 1 } }
6873     \dim_zero:N \extrarowheight

```

#4 is the optional argument of the command `\Block`, provided with the syntax `<...>`.

```

6874     #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.).

```

6875     \@@_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}`.

```

6876     \bool_if:NTF \l_@@_tabular_bool
6877     {
6878         \bool_lazy_all:nTF
6879         {
6880             { \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.

```

6881         { \dim_compare_p:n { \l_@@_col_width_dim >= \c_zero_dim } }
6882         { ! \g_@@_rotate_bool }
6883     }

```

When the block is mono-column in a column with a fixed width (eg `p{3cm}`), we use a `{minipage}`.

```

6884     {
6885         \use:x
6886         {
6887             \exp_not:N \begin { minipage }%
6888             [ \str_lowercase:V { \l_@@_vpos_of_block_str } ]
6889             { \l_@@_col_width_dim }
6890             \str_case:Vn \l_@@_hpos_block_str
6891             { c \centering r \raggedleft l \raggedright }
6892         }
6893         #5
6894     \end { minipage }
6895 }

```

In the other cases, we use a `{tabular}`.

```

6896     {
6897         \use:x
6898         {
6899             \exp_not:N \begin { tabular }%
6900             [ \str_lowercase:V { \l_@@_vpos_of_block_str } ]
6901             { @ { } \l_@@_hpos_block_str @ { } }
6902         }
6903         #5
6904     \end { tabular }
6905 }
6906 }

```

If we are in a mathematical array (`\l_@@_tabular_bool` is `false`). The composition is always done with an `{array}` (never with a `{minipage}`).

```

6907     {
6908         \c_math_toggle_token
6909         \use:x
6910         {
6911             \exp_not:N \begin { array }%
6912             [ \str_lowercase:V { \l_@@_vpos_of_block_str } ]
6913             { @ { } \l_@@_hpos_block_str @ { } }
6914         }
6915         #5
6916         \end { array }
6917         \c_math_toggle_token
6918     }
6919 }

```

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).

```

6920     \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.

```

6921     \int_compare:nNnT { #2 } = 1
6922     {
6923         \dim_gset:Nn \g_@@_blocks_wd_dim
6924         {
6925             \dim_max:nn
6926             \g_@@_blocks_wd_dim
6927             {
6928                 \box_wd:c
6929                 { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
6930             }
6931         }
6932     }

```

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.

```

6933     \int_compare:nNnT { #1 } = 1
6934     {
6935         \dim_gset:Nn \g_@@_blocks_ht_dim
6936         {
6937             \dim_max:nn
6938             \g_@@_blocks_ht_dim
6939             {
6940                 \box_ht:c
6941                 { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
6942             }
6943         }
6944         \dim_gset:Nn \g_@@_blocks_dp_dim
6945         {
6946             \dim_max:nn
6947             \g_@@_blocks_dp_dim
6948             {
6949                 \box_dp:c
6950                 { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
6951             }
6952         }
6953     }
6954     \seq_gput_right:Nx \g_@@_blocks_seq
6955     {
6956         \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.

```

6957     {
6958         \exp_not:n { #3 } ,
6959         \l_@@_hpos_block_str ,

```

Now, we put a key for the vertical alignment.

```

6960         \bool_if:NT \g_@@_rotate_bool
6961         {
6962             \bool_if:NTF \g_@@_rotate_c_bool
6963             { v-center }
6964             { \int_compare:nNnT \c@iRow = \l_@@_last_row_int T }
6965         }
6966     }
6967 }
6968 {
6969     \box_use_drop:c
6970     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
6971 }
6972 }
6973 \bool_set_false:N \g_@@_rotate_c_bool
6974 }

6975 \cs_new:Npn \@@_adjust_hpos_rotate:
6976 {
6977     \bool_if:NT \g_@@_rotate_bool
6978     {
6979         \str_set:Nx \l_@@_hpos_block_str
6980         {
6981             \bool_if:NTF \g_@@_rotate_c_bool
6982             { c }
6983             {
6984                 \str_case:VnF \l_@@_vpos_of_block_str
6985                 { b l B l t r T r }
6986                 { \int_compare:nNnTF \c@iRow = \l_@@_last_row_int r l }
6987             }
6988         }
6989     }
6990 }

```

Despite its name the following command rotates the box of the block *but also does vertical adjustement of the baseline of the block*.

```

6991 \cs_new_protected:Npn \@@_rotate_box_of_block:
6992 {
6993     \box_grotate:cn
6994     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
6995     { 90 }
6996     \int_compare:nNnT \c@iRow = \l_@@_last_row_int
6997     {
6998         \vbox_gset_top:cn
6999         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7000         {
7001             \skip_vertical:n { 0.8 ex }
7002             \box_use:c
7003             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7004         }
7005     }
7006     \bool_if:NT \g_@@_rotate_c_bool
7007     {
7008         \hbox_gset:cn

```

```

7009 { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7010 {
7011   \c_math_toggle_token
7012   \vcenter
7013   {
7014     \box_use:c
7015     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7016   }
7017   \c_math_toggle_token
7018 }
7019 }
7020 }

```

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.

```

7021 \cs_new_protected:Npn \@@_Block_v:nnnnn #1 #2 #3 #4 #5
7022 {
7023   \seq_gput_right:Nx \g_@@_blocks_seq
7024   {
7025     \l_tmpa_tl
7026     { \exp_not:n { #3 } }
7027     {
7028       \bool_if:NTF \l_@@_tabular_bool
7029       {
7030         \group_begin:
7031         \bool_if:NF \l_@@_respect_arraystretch_bool
7032         { \cs_set:Npn \exp_not:N \arraystretch { 1 } }
7033         \exp_not:n
7034         {
7035           \dim_zero:N \extrarowheight
7036           #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).

```

7037       % \@@_adjust_hpos_rotate:
7038       \use:x
7039       {
7040         \exp_not:N \begin { tabular } [ \l_@@_vpos_of_block_str ]
7041         { @ { } \l_@@_hpos_block_str @ { } }
7042       }
7043       #5
7044       \end { tabular }
7045     }
7046   \group_end:
7047 }

```

When we are *not* in an environments `{NiceTabular}` (or similar).

```

7048 {
7049   \group_begin:
7050   \bool_if:NF \l_@@_respect_arraystretch_bool
7051   { \cs_set:Npn \exp_not:N \arraystretch { 1 } }
7052   \exp_not:n
7053   {
7054     \dim_zero:N \extrarowheight
7055     #4

```

```

7056             % \@@_adjust_hpos_rotate:
7057             \c_math_toggle_token    % :n c
7058             \use:x
7059             {
7060                 \exp_not:N \begin { array } [ \l_@@_vpos_of_block_str ]
7061                 { @ { } \l_@@_hpos_block_str @ { } }
7062             }
7063             #5
7064             \end { array }
7065             \c_math_toggle_token
7066         }
7067     \group_end:
7068 }
7069 }
7070 }
7071 }

```

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).

```

7072 \keys_define:nn { NiceMatrix / Block / SecondPass }
7073 {
7074     tikz .code:n =
7075         \IfPackageLoadedTF { tikz }
7076         { \seq_put_right:Nn \l_@@_tikz_seq { { #1 } } }
7077         { \@@_error:n { tikz-key-without~tikz } } ,
7078     tikz .value_required:n = true ,
7079     fill .code:n =
7080         \tl_set_rescan:Nnn
7081         \l_@@_fill_tl
7082         { \char_set_catcode_other:N ! }
7083         { #1 } ,
7084     fill .value_required:n = true ,
7085     opacity .tl_set:N = \l_@@_opacity_tl ,
7086     opacity .value_required:n = true ,
7087     draw .code:n =
7088         \tl_set_rescan:Nnn
7089         \l_@@_draw_tl
7090         { \char_set_catcode_other:N ! }
7091         { #1 } ,
7092     draw .default:n = default ,
7093     rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
7094     rounded-corners .default:n = 4 pt ,
7095     color .code:n =
7096         \@@_color:n { #1 }
7097         \tl_set_rescan:Nnn
7098         \l_@@_draw_tl
7099         { \char_set_catcode_other:N ! }
7100         { #1 } ,
7101     borders .clist_set:N = \l_@@_borders_clist ,
7102     borders .value_required:n = true ,
7103     hvlines .meta:n = { vlines , hlines } ,
7104     vlines .bool_set:N = \l_@@_vlines_block_bool ,
7105     vlines .default:n = true ,
7106     hlines .bool_set:N = \l_@@_hlines_block_bool ,
7107     hlines .default:n = true ,
7108     line-width .dim_set:N = \l_@@_line_width_dim ,
7109     line-width .value_required:n = true ,

```

Some keys have not a property `.value_required:n` (or similar) because they are in `FirstPass`.

```

7110     l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7111     r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,

```

```

7112 c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7113 L .code:n = \str_set:Nn \l_@@_hpos_block_str l
7114         \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7115 R .code:n = \str_set:Nn \l_@@_hpos_block_str r
7116         \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7117 C .code:n = \str_set:Nn \l_@@_hpos_block_str c
7118         \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7119 t .code:n = \str_set:Nn \l_@@_vpos_of_block_str t ,
7120 T .code:n = \str_set:Nn \l_@@_vpos_of_block_str T ,
7121 b .code:n = \str_set:Nn \l_@@_vpos_of_block_str b ,
7122 B .code:n = \str_set:Nn \l_@@_vpos_of_block_str B ,
7123 v-center .code:n = \str_set:Nn \l_@@_vpos_of_block_str { c } ,
7124 v-center .value_forbidden:n = true ,
7125 name .tl_set:N = \l_@@_block_name_str ,
7126 name .value_required:n = true ,
7127 name .initial:n = ,
7128 respect-arraystretch .bool_set:N = \l_@@_respect_arraystretch_bool ,
7129 transparent .bool_set:N = \l_@@_transparent_bool ,
7130 transparent .default:n = true ,
7131 transparent .initial:n = false ,
7132 unknown .code:n = @@_error:n { Unknown~key~for~Block }
7133 }

```

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.

```

7134 \cs_new_protected:Npn \@@_draw_blocks:
7135 {
7136   \cs_set_eq:NN \ialign \@@_old_ialign:
7137   \seq_map_inline:Nn \g_@@_blocks_seq { \@@_Block_iv:nnnnnn ##1 }
7138 }
7139 \cs_new_protected:Npn \@@_Block_iv:nnnnnn #1 #2 #3 #4 #5 #6
7140 {

```

The integer `\l_@@_last_row_int` will be the last row of the block and `\l_@@_last_col_int` its last column.

```

7141   \int_zero_new:N \l_@@_last_row_int
7142   \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.

```

7143   \int_compare:nNnTF { #3 } > { 99 }
7144   { \int_set_eq:NN \l_@@_last_row_int \c{iRow }
7145     { \int_set:Nn \l_@@_last_row_int { #3 } }
7146   \int_compare:nNnTF { #4 } > { 99 }
7147   { \int_set_eq:NN \l_@@_last_col_int \c{jCol }
7148     { \int_set:Nn \l_@@_last_col_int { #4 } }
7149   \int_compare:nNnTF \l_@@_last_col_int > \g_@@_col_total_int
7150   {
7151     \int_compare:nTF
7152     { \l_@@_last_col_int <= \g_@@_static_num_of_col_int }
7153     {
7154       \msg_error:nnnn { nicematrix } { Block~too~large~2 } { #1 } { #2 }
7155       @@_msg_redirect_name:nn { Block~too~large~2 } { none }
7156       @@_msg_redirect_name:nn { columns~not~used } { none }
7157     }
7158     { \msg_error:nnnn { nicematrix } { Block~too~large~1 } { #1 } { #2 } }
7159   }

```

```

7160 {
7161   \int_compare:nNnTF \l_@@_last_row_int > \g_@@_row_total_int
7162     { \msg_error:nnnn { nicematrix } { Block~too~large~1 } { #1 } { #2 } }
7163     { \@@_Block_v:nnnnnn { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } }
7164   }
7165 }

```

The following command `\@@_Block_v:nnnnnn` will actually draw the block. #1 is the first row of the block; #2 is the first column of the block; #3 is the last row of the block; #4 is the last column of the block; #5 is a list of *key=value* options; #6 is the label

```

7166 \cs_new_protected:Npn \@@_Block_v:nnnnnn #1 #2 #3 #4 #5 #6
7167 {

```

The group is for the keys.

```

7168   \group_begin:
7169   \int_compare:nNnT { #1 } = { #3 }
7170     { \str_set:Nn \l_@@_vpos_of_block_str { t } }
7171   \keys_set:nn { NiceMatrix / Block / SecondPass } { #5 }
7172   \bool_if:NT \l_@@_vlines_block_bool
7173     {
7174       \tl_gput_right:Nx \g_nicematrix_code_after_tl
7175       {
7176         \@@_vlines_block:nnn
7177         { \exp_not:n { #5 } }
7178         { #1 - #2 }
7179         { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7180       }
7181     }
7182   \bool_if:NT \l_@@_hlines_block_bool
7183     {
7184       \tl_gput_right:Nx \g_nicematrix_code_after_tl
7185       {
7186         \@@_hlines_block:nnn
7187         { \exp_not:n { #5 } }
7188         { #1 - #2 }
7189         { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7190       }
7191     }
7192   \bool_if:nF
7193     {
7194       \l_@@_transparent_bool
7195       || ( \l_@@_vlines_block_bool && \l_@@_hlines_block_bool )
7196     }
7197   {

```

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).

```

7198     \seq_gput_left:Nx \g_@@_pos_of_blocks_seq
7199     { { #1 } { #2 } { #3 } { #4 } { \l_@@_block_name_str } }
7200   }

```

```

7201   \bool_lazy_and:nnT
7202     { ! ( \tl_if_empty_p:N \l_@@_draw_tl ) }
7203     { \l_@@_hlines_block_bool || \l_@@_vlines_block_bool }
7204     { \@@_error:n { hlines~with~color } }

```

```

7205   \tl_if_empty:NF \l_@@_draw_tl
7206     {
7207       \tl_gput_right:Nx \g_nicematrix_code_after_tl
7208       {
7209         \@@_stroke_block:nnn
7210         { \exp_not:n { #5 } } } % #5 are the options

```

```

7211         { #1 - #2 }
7212         { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7213     }
7214     \seq_gput_right:Nn \g_@@_pos_of_stroken_blocks_seq
7215     { { #1 } { #2 } { #3 } { #4 } }
7216 }
7217 \clist_if_empty:NF \l_@@_borders_clist
7218 {
7219     \tl_gput_right:Nx \g_nicematrix_code_after_tl
7220     {
7221         \@@_stroke_borders_block:nnn
7222         { \exp_not:n { #5 } }
7223         { #1 - #2 }
7224         { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7225     }
7226 }
7227 \tl_if_empty:NF \l_@@_fill_tl
7228 {
7229     \tl_if_empty:NF \l_@@_opacity_tl
7230     {
7231         \tl_if_head_eq_meaning:nNTF \l_@@_fill_tl [
7232             {
7233                 \tl_set:Nx \l_@@_fill_tl
7234                 {
7235                     [ opacity = \l_@@_opacity_tl ,
7236                     \tl_tail:V \l_@@_fill_tl
7237                 }
7238             }
7239             {
7240                 \tl_set:Nx \l_@@_fill_tl
7241                 { [ opacity = \l_@@_opacity_tl ] { \l_@@_fill_tl } }
7242             }
7243         ]
7244         \tl_gput_right:Nx \g_@@_pre_code_before_tl
7245         {
7246             \exp_not:N \roundedrectanglecolor
7247             \exp_args:NV \tl_if_head_eq_meaning:nNTF \l_@@_fill_tl [
7248                 { \l_@@_fill_tl }
7249                 { { \l_@@_fill_tl } }
7250             { #1 - #2 }
7251             { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7252             { \dim_use:N \l_@@_rounded_corners_dim }
7253         }
7254     }
7255 \seq_if_empty:NF \l_@@_tikz_seq
7256 {
7257     \tl_gput_right:Nx \g_nicematrix_code_before_tl
7258     {
7259         \@@_block_tikz:nnnnn
7260         { #1 }
7261         { #2 }
7262         { \int_use:N \l_@@_last_row_int }
7263         { \int_use:N \l_@@_last_col_int }
7264         { \seq_use:Nn \l_@@_tikz_seq { , } }
7265     }
7266 }
7267 \cs_set_protected_nopar:Npn \diagbox ##1 ##2
7268 {
7269     \tl_gput_right:Nx \g_@@_pre_code_after_tl
7270     {
7271         \@@_actually_diagbox:nnnnnn

```

```

7272         { #1 }
7273         { #2 }
7274         { \int_use:N \l_@@_last_row_int }
7275         { \int_use:N \l_@@_last_col_int }
7276         { \exp_not:n { ##1 } } { \exp_not:n { ##2 } }
7277     }
7278 }

7279 \hbox_set:Nn \l_@@_cell_box { \set@color #6 }
7280 \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
		two
three	four	five
six	seven	eight

We highlight the node `1-1-block-short`

our block		one
		two
three	four	five
six	seven	eight

The construction of the node corresponding to the merged cells.

```

7281 \pgfpicture
7282 \pgfrememberpicturepositiononpagetrue
7283 \pgf@relevantforpicturesizefalse
7284 \@@_qpoint:n { row - #1 }
7285 \dim_set_eq:NN \l_tmpa_dim \pgf@y
7286 \@@_qpoint:n { col - #2 }
7287 \dim_set_eq:NN \l_tmpb_dim \pgf@x
7288 \@@_qpoint:n { row - \int_eval:n { \l_@@_last_row_int + 1 } }
7289 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7290 \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7291 \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.

```

7292 \@@_pgf_rect_node:nnnnn
7293 { \@@_env: - #1 - #2 - block }
7294 \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
7295 \str_if_empty:NF \l_@@_block_name_str
7296 {
7297     \pgfnodealias
7298     { \@@_env: - \l_@@_block_name_str }
7299     { \@@_env: - #1 - #2 - block }
7300     \str_if_empty:NF \l_@@_name_str
7301     {
7302         \pgfnodealias
7303         { \l_@@_name_str - \l_@@_block_name_str }
7304         { \@@_env: - #1 - #2 - block }
7305     }
7306 }

```

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.

```

7307     \bool_if:NF \l_@@_hpos_of_block_cap_bool
7308     {
7309         \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.

```

7310         \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
7311         {

```

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.

```

7312             \cs_if_exist:cT
7313             { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
7314             {
7315                 \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
7316                 {
7317                     \pgfpointanchor { \@@_env: - ##1 - #2 } { west }
7318                     \dim_set:Nn \l_tmpb_dim { \dim_min:nn \l_tmpb_dim \pgf@x }
7319                 }
7320             }
7321         }

```

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.

```

7322         \dim_compare:nNnT \l_tmpb_dim = \c_max_dim
7323         {
7324             \@@_qpoint:n { col - #2 }
7325             \dim_set_eq:NN \l_tmpb_dim \pgf@x
7326         }
7327         \dim_set:Nn \l_@@_tmpd_dim { - \c_max_dim }
7328         \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
7329         {
7330             \cs_if_exist:cT
7331             { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
7332             {
7333                 \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
7334                 {
7335                     \pgfpointanchor
7336                     { \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
7337                     { east }
7338                     \dim_set:Nn \l_@@_tmpd_dim { \dim_max:nn \l_@@_tmpd_dim \pgf@x }
7339                 }
7340             }
7341         }
7342         \dim_compare:nNnT \l_@@_tmpd_dim = { - \c_max_dim }
7343         {
7344             \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7345             \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
7346         }
7347         \@@_pgf_rect_node:nnnnn
7348         { \@@_env: - #1 - #2 - block - short }
7349         \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
7350     }

```

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.

```

7351     \bool_if:NT \l_@@_medium_nodes_bool
7352     {
7353         \@@_pgf_rect_node:nnn

```

```

7354 { \@@_env: - #1 - #2 - block - medium }
7355 { \pgfpointanchor { \@@_env: - #1 - #2 - medium } { north-west } }
7356 {
7357     \pgfpointanchor
7358     { \@@_env:
7359         - \int_use:N \l_@@_last_row_int
7360         - \int_use:N \l_@@_last_col_int - medium
7361     }
7362     { south-east }
7363 }
7364 }

```

Now, we will put the label of the block.

```

7365 \bool_lazy_any:nTF
7366 {
7367     { \str_if_eq_p:Vn \l_@@_vpos_of_block_str { c } }
7368     { \str_if_eq_p:Vn \l_@@_vpos_of_block_str { T } }
7369     { \str_if_eq_p:Vn \l_@@_vpos_of_block_str { B } }
7370 }
7371 {

```

If we are in the first column, we must put the block as if it was with the key r.

```

7372     \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.

```

7373     \bool_if:nT \g_@@_last_col_found_bool
7374     {
7375         \int_compare:nNnT { #2 } = \g_@@_col_total_int
7376         { \str_set:Nn \l_@@_hpos_block_str l }
7377     }

```

\l_tmpa_tl will contain the anchor of the PGF node which will be used.

```

7378     \tl_set:Nx \l_tmpa_tl
7379     {
7380         \str_case:Vn \l_@@_vpos_of_block_str
7381         {
7382             c {
7383                 \str_case:Vn \l_@@_hpos_block_str
7384                 {
7385                     c { center }
7386                     l { west }
7387                     r { east }
7388                 }
7389             }
7390             T {
7391                 \str_case:Vn \l_@@_hpos_block_str
7392                 {
7393                     c { north }
7394                     l { north-west }
7395                     r { north-east }
7396                 }
7397             }
7398             }
7399             B {
7400                 \str_case:Vn \l_@@_hpos_block_str
7401                 {
7402                     c { south }
7403                     l { south-west }
7404                     r { south-east }
7405                 }
7406             }
7407         }
7408     }

```

```

7409     }
7410   }
7411   \pgftransformshift
7412   {
7413     \pgfpointanchor
7414     {
7415       \@@_env: - #1 - #2 - block
7416       \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
7417     }
7418     { \l_tmpa_tl }
7419   }
7420   \pgfset
7421   {
7422     inner~xsep = \c_zero_dim ,
7423     inner~ysep = \l_@@_block_ysep_dim
7424   }
7425   \pgfnode
7426   { rectangle }
7427   { \l_tmpa_tl }
7428   { \box_use_drop:N \l_@@_cell_box } { } { }
7429 }

```

End of the case when `\l_@@_vpos_of_block_str` is equal to c, T or B. Now, the other cases.

```

7430 {
7431   \pgfextracty \l_tmpa_dim
7432   {
7433     \@@_qpoint:n
7434     {
7435       row - \str_if_eq:NnTF \l_@@_vpos_of_block_str { b } { #3 } { #1 }
7436       - base
7437     }
7438   }
7439   \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.

```

7440   \pgfpointanchor
7441   {
7442     \@@_env: - #1 - #2 - block
7443     \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
7444   }
7445   {
7446     \str_case:Nn \l_@@_hpos_block_str
7447     {
7448       c { center }
7449       l { west }
7450       r { east }
7451     }
7452   }

```

We put the label of the block which has been composed in `\l_@@_cell_box`.

```

7453   \pgftransformshift { \pgfpoint \pgf@x \l_tmpa_dim }
7454   \pgfset { inner~sep = \c_zero_dim }
7455   \pgfnode
7456   { rectangle }
7457   {
7458     \str_case:Nn \l_@@_hpos_block_str
7459     {
7460       c { base }
7461       l { base~west }
7462       r { base~east }
7463     }
7464   }
7465   { \box_use_drop:N \l_@@_cell_box } { } { }
7466 }

```

```

7467 \endpgfpicture
7468 \group_end:
7469 }

```

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).

```

7470 \cs_new_protected:Npn \@@_stroke_block:nnn #1 #2 #3
7471 {
7472   \group_begin:
7473   \tl_clear:N \l_@@_draw_tl
7474   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
7475   \keys_set_known:nn { NiceMatrix / BlockStroke } { #1 }
7476   \pgfpicture
7477   \pgfrememberpicturepositiononpagetrue
7478   \pgf@relevantforpicturesizefalse
7479   \tl_if_empty:NF \l_@@_draw_tl
7480   {

```

If the user has used the key `color` of the command `\Block` without value, the color fixed by `\arrayrulecolor` is used.

```

7481     \str_if_eq:VnTF \l_@@_draw_tl { default }
7482     { \CT@arc@ }
7483     { \@@_color:V \l_@@_draw_tl }
7484   }
7485   \pgfsetcornersarced
7486   {
7487     \pgfpoint
7488     { \l_@@_rounded_corners_dim }
7489     { \l_@@_rounded_corners_dim }
7490   }
7491   \@@_cut_on_hyphen:w #2 \q_stop
7492   \bool_lazy_and:nnT
7493   { \int_compare_p:n { \l_tmpa_tl <= \c@iRow } }
7494   { \int_compare_p:n { \l_tmpb_tl <= \c@jCol } }
7495   {
7496     \@@_qpoint:n { row - \l_tmpa_tl }
7497     \dim_set_eq:NN \l_tmpb_dim \pgf@y
7498     \@@_qpoint:n { col - \l_tmpb_tl }
7499     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
7500     \@@_cut_on_hyphen:w #3 \q_stop
7501     \int_compare:nNnT \l_tmpa_tl > \c@iRow
7502     { \tl_set:Nx \l_tmpa_tl { \int_use:N \c@iRow } }
7503     \int_compare:nNnT \l_tmpb_tl > \c@jCol
7504     { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@jCol } }
7505     \@@_qpoint:n { row - \int_eval:n { \l_tmpa_tl + 1 } }
7506     \dim_set_eq:NN \l_tmpa_dim \pgf@y
7507     \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
7508     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
7509     \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
7510     \pgfpathrectanglecorners
7511     { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
7512     { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
7513     \dim_compare:nNnTF \l_@@_rounded_corners_dim = \c_zero_dim
7514     { \pgfusepathqstroke }
7515     { \pgfusepath { stroke } }
7516   }
7517   \endpgfpicture
7518   \group_end:
7519 }

```

Here is the set of keys for the command `\@@_stroke_block:nnn`.

```

7520 \keys_define:nn { NiceMatrix / BlockStroke }

```

```

7521 {
7522   color .tl_set:N = \l_@@_draw_tl ,
7523   draw .code:n =
7524     \exp_args:Nx \tl_if_empty:nF { #1 } { \tl_set:Nn \l_@@_draw_tl { #1 } } ,
7525   draw .default:n = default ,
7526   line-width .dim_set:N = \l_@@_line_width_dim ,
7527   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
7528   rounded-corners .default:n = 4 pt
7529 }

```

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).

```

7530 \cs_new_protected:Npn \@@_vlines_block:nnn #1 #2 #3
7531 {
7532   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
7533   \keys_set_known:nn { NiceMatrix / BlockBorders } { #1 }
7534   \@@_cut_on_hyphen:w #2 \q_stop
7535   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
7536   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
7537   \@@_cut_on_hyphen:w #3 \q_stop
7538   \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
7539   \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
7540   \int_step_inline:nnn \l_@@_tmpd_tl \l_tmpb_tl
7541   {
7542     \use:x
7543     {
7544       \@@_vline:n
7545       {
7546         position = ##1 ,
7547         start = \l_@@_tmpc_tl ,
7548         end = \int_eval:n { \l_tmpa_tl - 1 } ,
7549         total-width = \dim_use:N \l_@@_line_width_dim % added 2022-08-06
7550       }
7551     }
7552   }
7553 }
7554 \cs_new_protected:Npn \@@_hlines_block:nnn #1 #2 #3
7555 {
7556   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
7557   \keys_set_known:nn { NiceMatrix / BlockBorders } { #1 }
7558   \@@_cut_on_hyphen:w #2 \q_stop
7559   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
7560   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
7561   \@@_cut_on_hyphen:w #3 \q_stop
7562   \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
7563   \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
7564   \int_step_inline:nnn \l_@@_tmpc_tl \l_tmpa_tl
7565   {
7566     \use:x
7567     {
7568       \@@_hline:n
7569       {
7570         position = ##1 ,
7571         start = \l_@@_tmpd_tl ,
7572         end = \int_eval:n { \l_tmpb_tl - 1 } ,
7573         total-width = \dim_use:N \l_@@_line_width_dim
7574       }
7575     }
7576   }
7577 }

```

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).

```

7578 \cs_new_protected:Npn \@@_stroke_borders_block:nnn #1 #2 #3
7579 {
7580   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
7581   \keys_set_known:nn { NiceMatrix / BlockBorders } { #1 }
7582   \dim_compare:nNnTF \l_@@_rounded_corners_dim > \c_zero_dim
7583     { \@@_error:n { borders~forbidden } }
7584     {
7585       \tl_clear_new:N \l_@@_borders_tikz_tl
7586       \keys_set:nV
7587         { NiceMatrix / OnlyForTikzInBorders }
7588         \l_@@_borders_clist
7589       \@@_cut_on_hyphen:w #2 \q_stop
7590       \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
7591       \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
7592       \@@_cut_on_hyphen:w #3 \q_stop
7593       \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
7594       \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
7595       \@@_stroke_borders_block_i:
7596     }
7597 }
7598 \hook_gput_code:nnn { begindocument } { . }
7599 {
7600   \cs_new_protected:Npx \@@_stroke_borders_block_i:
7601     {
7602       \c_@@_pgfortikzpicture_tl
7603       \@@_stroke_borders_block_ii:
7604       \c_@@_endpgfortikzpicture_tl
7605     }
7606 }
7607 \cs_new_protected:Npn \@@_stroke_borders_block_ii:
7608 {
7609   \pgfrememberpicturepositiononpagetrue
7610   \pgf@relevantforpicturesizefalse
7611   \CT@arc@
7612   \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
7613   \clist_if_in:NnT \l_@@_borders_clist { right }
7614     { \@@_stroke_vertical:n \l_tmpb_tl }
7615   \clist_if_in:NnT \l_@@_borders_clist { left }
7616     { \@@_stroke_vertical:n \l_@@_tmpd_tl }
7617   \clist_if_in:NnT \l_@@_borders_clist { bottom }
7618     { \@@_stroke_horizontal:n \l_tmpa_tl }
7619   \clist_if_in:NnT \l_@@_borders_clist { top }
7620     { \@@_stroke_horizontal:n \l_@@_tmpc_tl }
7621 }
7622 \keys_define:nn { NiceMatrix / OnlyForTikzInBorders }
7623 {
7624   tikz .code:n =
7625     \cs_if_exist:NTF \tikzpicture
7626       { \tl_set:Nn \l_@@_borders_tikz_tl { #1 } }
7627       { \@@_error:n { tikz-in~borders~without-tikz } } ,
7628   tikz .value_required:n = true ,
7629   top .code:n = ,
7630   bottom .code:n = ,
7631   left .code:n = ,
7632   right .code:n = ,
7633   unknown .code:n = \@@_error:n { bad~border }
7634 }

```

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).

```

7635 \cs_new_protected:Npn \@@_stroke_vertical:n #1
7636 {
7637   \@@_qpoint:n \l_@@_tmpc_tl
7638   \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
7639   \@@_qpoint:n \l_tmpa_tl
7640   \dim_set:Nn \l_@@_tmpc_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
7641   \@@_qpoint:n { #1 }
7642   \tl_if_empty:NTF \l_@@_borders_tikz_tl
7643   {
7644     \pgfpathmoveto { \pgfpoint \pgf@x \l_tmpb_dim }
7645     \pgfpathlineto { \pgfpoint \pgf@x \l_@@_tmpc_dim }
7646     \pgfusepathqstroke
7647   }
7648   {
7649     \use:x { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
7650     ( \pgf@x , \l_tmpb_dim ) -- ( \pgf@x , \l_@@_tmpc_dim ) ;
7651   }
7652 }

```

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).

```

7653 \cs_new_protected:Npn \@@_stroke_horizontal:n #1
7654 {
7655   \@@_qpoint:n \l_@@_tmpd_tl
7656   \clist_if_in:NnTF \l_@@_borders_clist { left }
7657   { \dim_set:Nn \l_tmpa_dim { \pgf@x - 0.5 \l_@@_line_width_dim } }
7658   { \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \l_@@_line_width_dim } }
7659   \@@_qpoint:n \l_tmpb_tl
7660   \dim_set:Nn \l_tmpb_dim { \pgf@x + 0.5 \l_@@_line_width_dim }
7661   \@@_qpoint:n { #1 }
7662   \tl_if_empty:NTF \l_@@_borders_tikz_tl
7663   {
7664     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }
7665     \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
7666     \pgfusepathqstroke
7667   }
7668   {
7669     \use:x { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
7670     ( \l_tmpa_dim , \pgf@y ) -- ( \l_tmpb_dim , \pgf@y ) ;
7671   }
7672 }

```

Here is the set of keys for the command \@@_stroke_borders_block:nnn.

```

7673 \keys_define:nn { NiceMatrix / BlockBorders }
7674 {
7675   borders .clist_set:N = \l_@@_borders_clist ,
7676   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
7677   rounded-corners .default:n = 4 pt ,
7678   line-width .dim_set:N = \l_@@_line_width_dim ,
7679 }

```

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.

```

7680 \cs_new_protected:Npn \@@_block_tikz:nnnnn #1 #2 #3 #4 #5
7681 {
7682   \begin { tikzpicture }
7683   \@@_clip_with_rounded_corners:
7684   \clist_map_inline:nn { #5 }
7685   {
7686     \path [ ##1 ]
7687     ( #1 -| #2 )

```

```

7688         rectangle
7689         ( \int_eval:n { #3 + 1 } -| \int_eval:n { #4 + 1 } ) ;
7690     }
7691 \end { tikzpicture }
7692 }

```

28 How to draw the dotted lines transparently

```

7693 \cs_set_protected:Npn \@@_renew_matrix:
7694 {
7695     \RenewDocumentEnvironment { pmatrix } { }
7696     { \pNiceMatrix }
7697     { \endpNiceMatrix }
7698     \RenewDocumentEnvironment { vmatrix } { }
7699     { \vNiceMatrix }
7700     { \endvNiceMatrix }
7701     \RenewDocumentEnvironment { Vmatrix } { }
7702     { \VNiceMatrix }
7703     { \endVNiceMatrix }
7704     \RenewDocumentEnvironment { bmatrix } { }
7705     { \bNiceMatrix }
7706     { \endbNiceMatrix }
7707     \RenewDocumentEnvironment { Bmatrix } { }
7708     { \BNiceMatrix }
7709     { \endBNiceMatrix }
7710 }

```

29 Automatic arrays

We will extract some keys and pass the other keys to the environment `{NiceArrayWithDelims}`.

```

7711 \keys_define:nn { NiceMatrix / Auto }
7712 {
7713     columns-type .code:n = \@@_set_preamble:Nn \l_@@_columns_type_tl { #1 } ,
7714     columns-type .value_required:n = true ,
7715     l .meta:n = { columns-type = l } ,
7716     r .meta:n = { columns-type = r } ,
7717     c .meta:n = { columns-type = c } ,
7718     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
7719     delimiters / color .value_required:n = true ,
7720     delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
7721     delimiters / max-width .default:n = true ,
7722     delimiters .code:n = \keys_set:nn { NiceMatrix / delimiters } { #1 } ,
7723     delimiters .value_required:n = true ,
7724     rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
7725     rounded-corners .default:n = 4 pt
7726 }
7727 \NewDocumentCommand \AutoNiceMatrixWithDelims
7728 { m m O { } } > { \SplitArgument { 1 } { - } } m O { } m ! O { } }
7729 { \@@_auto_nice_matrix:nnnnnn { #1 } { #2 } #4 { #6 } { #3 , #5 , #7 } }
7730 \cs_new_protected:Npn \@@_auto_nice_matrix:nnnnnn #1 #2 #3 #4 #5 #6
7731 {

```

The group is for the protection of the keys.

```

7732 \group_begin:
7733 \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).

```

7734 \bool_set_false:N \l_@@_preamble_bool
7735 \use:x
7736 {
7737   \exp_not:N \begin { NiceArrayWithDelims } { #1 } { #2 }
7738   { * { #4 } { \exp_not:V \l_@@_columns_type_tl } }
7739   [ \exp_not:V \l_tmpa_tl ]
7740 }
7741 \int_compare:nNnT \l_@@_first_row_int = 0
7742 {
7743   \int_compare:nNnT \l_@@_first_col_int = 0 { & }
7744   \prg_replicate:nn { #4 - 1 } { & }
7745   \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
7746 }
7747 \prg_replicate:nn { #3 }
7748 {
7749   \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`).

```

7750   \prg_replicate:nn { #4 - 1 } { { } #5 & } #5
7751   \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
7752 }
7753 \int_compare:nNnT \l_@@_last_row_int > { -2 }
7754 {
7755   \int_compare:nNnT \l_@@_first_col_int = 0 { & }
7756   \prg_replicate:nn { #4 - 1 } { & }
7757   \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
7758 }
7759 \end { NiceArrayWithDelims }
7760 \group_end:
7761 }

7762 \cs_set_protected:Npn \@@_define_com:nnn #1 #2 #3
7763 {
7764   \cs_set_protected:cpn { #1 AutoNiceMatrix }
7765   {
7766     \bool_gset_true:N \g_@@_delims_bool
7767     \str_gset:Nx \g_@@_name_env_str { #1 AutoNiceMatrix }
7768     \AutoNiceMatrixWithDelims { #2 } { #3 }
7769   }
7770 }

7771 \@@_define_com:nnn p ( )
7772 \@@_define_com:nnn b [ ]
7773 \@@_define_com:nnn v | |
7774 \@@_define_com:nnn V \ | \ |
7775 \@@_define_com:nnn B \{ \}

```

We define also a command `\AutoNiceMatrix` similar to the environment `{NiceMatrix}`.

```

7776 \NewDocumentCommand \AutoNiceMatrix { 0 { } m 0 { } m ! 0 { } }
7777 {
7778   \group_begin:
7779   \bool_gset_false:N \g_@@_delims_bool
7780   \AutoNiceMatrixWithDelims . . { #2 } { #4 } [ #1 , #3 , #5 ]
7781   \group_end:
7782 }

```

30 The redefinition of the command `\dotfill`

```

7783 \cs_set_eq:NN \@@_old_dotfill \dotfill
7784 \cs_new_protected:Npn \@@_dotfill:
7785 {

```

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}`).

```

7786   \@@_old_dotfill
7787   \tl_gput_right:Nn \g_@@_cell_after_hook_tl \@@_dotfill_i:
7788 }

```

Now, if the box is not empty (unfortunately, we can’t actually test whether the box is empty and that’s why we only consider its width), we insert `\@@_dotfill` (which is the saved version of `\dotfill`) in the cell of the array, and it will extend, since it is no longer in `\l_@@_cell_box`.

```

7789 \cs_new_protected:Npn \@@_dotfill_i:
7790 { \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.

```

7791 \cs_new_protected:Npn \@@_diagbox:nn #1 #2
7792 {
7793   \tl_gput_right:Nx \g_@@_pre_code_after_tl
7794   {
7795     \@@_actually_diagbox:nnnnnn
7796     { \int_use:N \c@iRow }
7797     { \int_use:N \c@jCol }
7798     { \int_use:N \c@iRow }
7799     { \int_use:N \c@jCol }
7800     { \exp_not:n { #1 } }
7801     { \exp_not:n { #2 } }
7802   }

```

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`.

```

7803   \seq_gput_right:Nx \g_@@_pos_of_blocks_seq
7804   {
7805     { \int_use:N \c@iRow }
7806     { \int_use:N \c@jCol }
7807     { \int_use:N \c@iRow }
7808     { \int_use:N \c@jCol }

```

The last argument is for the name of the block.

```

7809     { }
7810   }
7811 }

```

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.

```

7812 \cs_new_protected:Npn \@@_actually_diagbox:nnnnnn #1 #2 #3 #4 #5 #6
7813 {
7814   \pgfpicture
7815   \pgf@relevantforpicturesizefalse
7816   \pgfrememberpicturepositiononpagetrue
7817   \@@_qpoint:n { row - #1 }
7818   \dim_set_eq:NN \l_tmpa_dim \pgf@y
7819   \@@_qpoint:n { col - #2 }

```

```

7820 \dim_set_eq:NN \l_tmpb_dim \pgf@x
7821 \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
7822 \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
7823 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7824 \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
7825 \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
7826 \pgfpathlineto { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
7827 {

```

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.

```

7828 \CT@arc@
7829 \pgfsetroundcap
7830 \pgfusepathqstroke
7831 }
7832 \pgfset { inner~sep = 1 pt }
7833 \pgfscope
7834 \pgftransformshift { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
7835 \pgfnode { rectangle } { south-west }
7836 {
7837 \begin { minipage } { 20 cm }
7838 \@@_math_toggle_token: #5 \@@_math_toggle_token:
7839 \end { minipage }
7840 }
7841 { }
7842 { }
7843 \endpgfscope
7844 \pgftransformshift { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
7845 \pgfnode { rectangle } { north-east }
7846 {
7847 \begin { minipage } { 20 cm }
7848 \raggedleft
7849 \@@_math_toggle_token: #6 \@@_math_toggle_token:
7850 \end { minipage }
7851 }
7852 { }
7853 { }
7854 \endpgfpicture
7855 }

```

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.

```

7856 \keys_define:nn { NiceMatrix }
7857 {
7858 \CodeAfter / rules .inherit:n = NiceMatrix / rules ,
7859 \CodeAfter / sub-matrix .inherit:n = NiceMatrix / sub-matrix
7860 }
7861 \keys_define:nn { NiceMatrix / CodeAfter }
7862 {
7863 sub-matrix .code:n = \keys_set:nn { NiceMatrix / sub-matrix } { #1 } ,
7864 sub-matrix .value_required:n = true ,
7865 delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
7866 delimiters / color .value_required:n = true ,
7867 rules .code:n = \keys_set:nn { NiceMatrix / rules } { #1 } ,
7868 rules .value_required:n = true ,
7869 xdots .code:n = \keys_set:nn { NiceMatrix / xdots } { #1 } ,
7870 unknown .code:n = \@@_error:n { Unknown-key-for-CodeAfter }
7871 }

```

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`.

```
7872 \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 `\`.

```
7873 \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`.

```
7874 \cs_new_protected:Npn \@@_CodeAfter_ii:n #1 \end
7875 {
7876   \tl_gput_right:Nn \g_nicematrix_code_after_tl { #1 }
7877   \@@_CodeAfter_iv:n
7878 }
```

We catch the argument of the command `\end` (in `#1`).

```
7879 \cs_new_protected:Npn \@@_CodeAfter_iv:n #1
7880 {
```

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.

```
7881   \str_if_eq:eeTF \@currenvir { #1 }
7882   { \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`.

```
7883   {
7884     \tl_gput_right:Nn \g_nicematrix_code_after_tl { \end { #1 } }
7885     \@@_CodeAfter_ii:n
7886   }
7887 }
```

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.

```
7888 \cs_new_protected:Npn \@@_delimiter:nnn #1 #2 #3
7889 {
7890   \pgfpicture
7891   \pgfrememberpicturepositiononpagetrue
7892   \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.

```

7893 \@@_qpoint:n { row - 1 }
7894 \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
7895 \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
7896 \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).

```

7897 \bool_if:nTF { #3 }
7898 { \dim_set_eq:NN \l_tmpa_dim \c_max_dim }
7899 { \dim_set:Nn \l_tmpa_dim { - \c_max_dim } }
7900 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
7901 {
7902   \cs_if_exist:cT
7903   { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
7904   {
7905     \pgfpointanchor
7906     { \@@_env: - ##1 - #2 }
7907     { \bool_if:nTF { #3 } { west } { east } }
7908     \dim_set:Nn \l_tmpa_dim
7909     { \bool_if:nTF { #3 } \dim_min:nn \dim_max:nn \l_tmpa_dim \pgf@x }
7910   }
7911 }

```

Now we can put the delimiter with a node of PGF.

```

7912 \pgfset { inner~sep = \c_zero_dim }
7913 \dim_zero:N \nulldelimiterspace
7914 \pgftransformshift
7915 {
7916   \pgfpoint
7917   { \l_tmpa_dim }
7918   { ( \l_@@_y_initial_dim + \l_@@_y_final_dim + \arrayrulewidth ) / 2 }
7919 }
7920 \pgfnode
7921 { rectangle }
7922 { \bool_if:nTF { #3 } { east } { west } }
7923 {

```

Here is the content of the PGF node, that is to say the delimiter, constructed with its right size.

```

7924 \nullfont
7925 \c_math_toggle_token
7926 \@@_color:V \l_@@_delimiters_color_tl
7927 \bool_if:nTF { #3 } { \left #1 } { \left . }
7928 \vcenter
7929 {
7930   \nullfont
7931   \hrule \@height
7932   \dim_eval:n { \l_@@_y_initial_dim - \l_@@_y_final_dim }
7933   \@depth \c_zero_dim
7934   \@width \c_zero_dim
7935 }
7936 \bool_if:nTF { #3 } { \right . } { \right #1 }
7937 \c_math_toggle_token
7938 }
7939 { }
7940 { }
7941 \endpgfpicture
7942 }

```

34 The command \SubMatrix

```

7943 \keys_define:nn { NiceMatrix / sub-matrix }
7944 {
7945   extra-height .dim_set:N = \l_@@_submatrix_extra_height_dim ,
7946   extra-height .value_required:n = true ,
7947   left-xshift .dim_set:N = \l_@@_submatrix_left_xshift_dim ,
7948   left-xshift .value_required:n = true ,
7949   right-xshift .dim_set:N = \l_@@_submatrix_right_xshift_dim ,
7950   right-xshift .value_required:n = true ,
7951   xshift .meta:n = { left-xshift = #1, right-xshift = #1 } ,
7952   xshift .value_required:n = true ,
7953   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
7954   delimiters / color .value_required:n = true ,
7955   slim .bool_set:N = \l_@@_submatrix_slim_bool ,
7956   slim .default:n = true ,
7957   hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
7958   hlines .default:n = all ,
7959   vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
7960   vlines .default:n = all ,
7961   hvlines .meta:n = { hlines, vlines } ,
7962   hvlines .value_forbidden:n = true ,
7963 }
7964 \keys_define:nn { NiceMatrix }
7965 {
7966   SubMatrix .inherit:n = NiceMatrix / sub-matrix ,
7967   CodeAfter / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
7968   NiceMatrix / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
7969   NiceArray / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
7970   pNiceArray / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
7971   NiceMatrixOptions / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
7972 }

```

The following keys set is for the command \SubMatrix itself (not the tuning of \SubMatrix that can be done elsewhere).

```

7973 \keys_define:nn { NiceMatrix / SubMatrix }
7974 {
7975   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
7976   delimiters / color .value_required:n = true ,
7977   hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
7978   hlines .default:n = all ,
7979   vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
7980   vlines .default:n = all ,
7981   hvlines .meta:n = { hlines, vlines } ,
7982   hvlines .value_forbidden:n = true ,
7983   name .code:n =
7984     \tl_if_empty:nTF { #1 }
7985     { \@@_error:n { Invalid-name } }
7986     {
7987       \regex_match:nnTF { \A[A-Za-z][A-Za-z0-9]*\Z } { #1 }
7988       {
7989         \seq_if_in:NnTF \g_@@_submatrix_names_seq { #1 }
7990         { \@@_error:nn { Duplicate-name-for-SubMatrix } { #1 } }
7991         {
7992           \str_set:Nn \l_@@_submatrix_name_str { #1 }
7993           \seq_gput_right:Nn \g_@@_submatrix_names_seq { #1 }
7994         }
7995       }
7996       { \@@_error:n { Invalid-name } }
7997     } ,
7998   name .value_required:n = true ,
7999   rules .code:n = \keys_set:nn { NiceMatrix / rules } { #1 } ,

```

```

8000 rules .value_required:n = true ,
8001 code .tl_set:N = \l_@@_code_tl ,
8002 code .value_required:n = true ,
8003 unknown .code:n = \@@_error:n { Unknown~key~for~SubMatrix }
8004 }

8005 \NewDocumentCommand \@@_SubMatrix_in_code_before { m m m m ! 0 { } }
8006 {
8007   \peek_remove_spaces:n
8008   {
8009     \tl_gput_right:Nx \g_@@_pre_code_after_tl
8010     {
8011       \SubMatrix { #1 } { #2 } { #3 } { #4 }
8012       [
8013         delimiters / color = \l_@@_delimiters_color_tl ,
8014         hlines = \l_@@_submatrix_hlines_clist ,
8015         vlines = \l_@@_submatrix_vlines_clist ,
8016         extra-height = \dim_use:N \l_@@_submatrix_extra_height_dim ,
8017         left-xshift = \dim_use:N \l_@@_submatrix_left_xshift_dim ,
8018         right-xshift = \dim_use:N \l_@@_submatrix_right_xshift_dim ,
8019         slim = \bool_to_str:N \l_@@_submatrix_slim_bool ,
8020         #5
8021       ]
8022     }
8023     \@@_SubMatrix_in_code_before_i { #2 } { #3 }
8024   }
8025 }

8026 \NewDocumentCommand \@@_SubMatrix_in_code_before_i
8027 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
8028 { \@@_SubMatrix_in_code_before_i:nnnn #1 #2 }

8029 \cs_new_protected:Npn \@@_SubMatrix_in_code_before_i:nnnn #1 #2 #3 #4
8030 {
8031   \seq_gput_right:Nx \g_@@_submatrix_seq
8032   {
We use \str_if_eq:nnTF because it is fully expandable.
8033     { \str_if_eq:nnTF { #1 } { last } { \int_use:N \c@iRow } { #1 } }
8034     { \str_if_eq:nnTF { #2 } { last } { \int_use:N \c@jCol } { #2 } }
8035     { \str_if_eq:nnTF { #3 } { last } { \int_use:N \c@iRow } { #3 } }
8036     { \str_if_eq:nnTF { #4 } { last } { \int_use:N \c@jCol } { #4 } }
8037   }
8038 }

```

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.

```

8039 \hook_gput_code:nnn { begindocument } { . }
8040 {

```

```

8041 \tl_set:Nn \l_@@_argspec_tl { m m m m 0 { } E { _ ^ } { { } { } } }
8042 \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
8043 \exp_args:NNV \NewDocumentCommand \@@_SubMatrix \l_@@_argspec_tl
8044 {
8045   \peek_remove_spaces:n
8046   {
8047     \@@_sub_matrix:nnnnnnn
8048     { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } { #7 }
8049   }
8050 }
8051 }

```

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).

```

8052 \NewDocumentCommand \@@_compute_i_j:nn
8053 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
8054 { \@@_compute_i_j:nnnn #1 #2 }

8055 \cs_new_protected:Npn \@@_compute_i_j:nnnn #1 #2 #3 #4
8056 {
8057   \tl_set:Nn \l_@@_first_i_tl { #1 }
8058   \tl_set:Nn \l_@@_first_j_tl { #2 }
8059   \tl_set:Nn \l_@@_last_i_tl { #3 }
8060   \tl_set:Nn \l_@@_last_j_tl { #4 }
8061   \tl_if_eq:NnT \l_@@_first_i_tl { last }
8062   { \tl_set:NV \l_@@_first_i_tl \c@iRow }
8063   \tl_if_eq:NnT \l_@@_first_j_tl { last }
8064   { \tl_set:NV \l_@@_first_j_tl \c@jCol }
8065   \tl_if_eq:NnT \l_@@_last_i_tl { last }
8066   { \tl_set:NV \l_@@_last_i_tl \c@iRow }
8067   \tl_if_eq:NnT \l_@@_last_j_tl { last }
8068   { \tl_set:NV \l_@@_last_j_tl \c@jCol }
8069 }

8070 \cs_new_protected:Npn \@@_sub_matrix:nnnnnnn #1 #2 #3 #4 #5 #6 #7
8071 {
8072   \group_begin:

```

The four following token lists correspond to the position of the `\SubMatrix`.

```

8073 \@@_compute_i_j:nn { #2 } { #3 }
8074 % added 6.19b
8075 \int_compare:nNnT \l_@@_first_i_tl = \l_@@_last_i_tl
8076 { \cs_set:Npn \arraystretch { 1 } }
8077 \bool_lazy_or:nnTF
8078 { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
8079 { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
8080 { \@@_error:nn { Construct~too~large } { \SubMatrix } }
8081 {
8082   \str_clear_new:N \l_@@_submatrix_name_str
8083   \keys_set:nn { NiceMatrix / SubMatrix } { #5 }
8084   \pgfpicture
8085   \pgfrememberpicturepositiononpagetrue
8086   \pgf@relevantforpicturesizefalse
8087   \pgfset { inner~sep = \c_zero_dim }
8088   \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
8089   \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }

```

The last value of `\int_step_inline:nnn` is provided by currification.

```

8090 \bool_if:NTF \l_@@_submatrix_slim_bool
8091 { \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl }
8092 { \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int }
8093 {
8094   \cs_if_exist:cT
8095   { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }

```

```

8096         {
8097             \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
8098             \dim_set:Nn \l_@@_x_initial_dim
8099                 { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
8100         }
8101     \cs_if_exist:cT
8102     { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
8103     {
8104         \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
8105         \dim_set:Nn \l_@@_x_final_dim
8106             { \dim_max:nn \l_@@_x_final_dim \pgf@x }
8107     }
8108 }
8109 \dim_compare:nNnTF \l_@@_x_initial_dim = \c_max_dim
8110 { \@@_error:nn { Impossible-delimiter } { left } }
8111 {
8112     \dim_compare:nNnTF \l_@@_x_final_dim = { - \c_max_dim }
8113     { \@@_error:nn { Impossible-delimiter } { right } }
8114     { \@@_sub_matrix_i:nnnn { #1 } { #4 } { #6 } { #7 } }
8115 }
8116 \endpgfpicture
8117 }
8118 \group_end:
8119 }

```

#1 is the left delimiter, #2 is the right one, #3 is the subscript and #4 is the superscript.

```

8120 \cs_new_protected:Npn \@@_sub_matrix_i:nnnn #1 #2 #3 #4
8121 {
8122     \@@_qpoint:n { row - \l_@@_first_i_tl - base }
8123     \dim_set:Nn \l_@@_y_initial_dim
8124     {
8125         \fp_to_dim:n
8126         {
8127             \pgf@y
8128             + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
8129         }
8130     } % modified 6.13c
8131     \@@_qpoint:n { row - \l_@@_last_i_tl - base }
8132     \dim_set:Nn \l_@@_y_final_dim
8133     { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
8134     % modified 6.13c
8135     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
8136     {
8137         \cs_if_exist:cT
8138         { pgf @ sh @ ns @ \@@_env: - \l_@@_first_i_tl - ##1 }
8139         {
8140             \pgfpointanchor { \@@_env: - \l_@@_first_i_tl - ##1 } { north }
8141             \dim_set:Nn \l_@@_y_initial_dim
8142                 { \dim_max:nn \l_@@_y_initial_dim \pgf@y }
8143         }
8144         \cs_if_exist:cT
8145         { pgf @ sh @ ns @ \@@_env: - \l_@@_last_i_tl - ##1 }
8146         {
8147             \pgfpointanchor { \@@_env: - \l_@@_last_i_tl - ##1 } { south }
8148             \dim_set:Nn \l_@@_y_final_dim
8149                 { \dim_min:nn \l_@@_y_final_dim \pgf@y }
8150         }
8151     }
8152     \dim_set:Nn \l_tmpa_dim
8153     {
8154         \l_@@_y_initial_dim - \l_@@_y_final_dim +
8155         \l_@@_submatrix_extra_height_dim - \arrayrulewidth
8156     }

```

```
8157 \dim_zero:N \nulldelimiterspace
```

We will draw the rules in the `\SubMatrix`.

```
8158 \group_begin:
8159 \pgfsetlinewidth { 1.1 \arrayrulewidth }
8160 \@@_set_CT@arc@:V \l_@@_rules_color_tl
8161 \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`.

```
8162 \seq_map_inline:Nn \g_@@_cols_vlism_seq
8163 {
8164   \int_compare:nNt \l_@@_first_j_tl < { ##1 }
8165   {
8166     \int_compare:nNt
8167       { ##1 } < { \int_eval:n { \l_@@_last_j_tl + 1 } }
8168     {
```

First, we extract the value of the abscissa of the rule we have to draw.

```
8169       \@@_qpoint:n { col - ##1 }
8170       \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
8171       \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
8172       \pgfusepathqstroke
8173     }
8174   }
8175 }
```

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.

```
8176 \tl_if_eq:NnTF \l_@@_submatrix_vlines_clist { all }
8177 { \int_step_inline:nn { \l_@@_last_j_tl - \l_@@_first_j_tl } }
8178 { \clist_map_inline:Nn \l_@@_submatrix_vlines_clist }
8179 {
8180   \bool_lazy_and:nnTF
8181     { \int_compare_p:nNn { ##1 } > 0 }
8182     {
8183       \int_compare_p:nNn
8184         { ##1 } < { \l_@@_last_j_tl - \l_@@_first_j_tl + 1 } }
8185       {
8186         \@@_qpoint:n { col - \int_eval:n { ##1 + \l_@@_first_j_tl } }
8187         \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
8188         \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
8189         \pgfusepathqstroke
8190       }
8191       { \@@_error:nnn { Wrong~line~in~SubMatrix } { vertical } { ##1 } }
8192     }
8193 }
```

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.

```
8193 \tl_if_eq:NnTF \l_@@_submatrix_hlines_clist { all }
8194 { \int_step_inline:nn { \l_@@_last_i_tl - \l_@@_first_i_tl } }
8195 { \clist_map_inline:Nn \l_@@_submatrix_hlines_clist }
8196 {
8197   \bool_lazy_and:nnTF
8198     { \int_compare_p:nNn { ##1 } > 0 }
8199     {
8200       \int_compare_p:nNn
8201         { ##1 } < { \l_@@_last_i_tl - \l_@@_first_i_tl + 1 } }
8202       {
8203         \@@_qpoint:n { row - \int_eval:n { ##1 + \l_@@_first_i_tl } }
8204
```

We use a group to protect `\l_tmpa_dim` and `\l_tmpb_dim`.

```
8204 \group_begin:
```

We compute in `\l_tmpa_dim` the x -value of the left end of the rule.

```

8205     \dim_set:Nn \l_tmpa_dim
8206     { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
8207     \str_case:nn { #1 }
8208     {
8209         ( { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
8210         [ { \dim_sub:Nn \l_tmpa_dim { 0.2 mm } }
8211         \{ { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
8212         }
8213     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }

```

We compute in `\l_tmpb_dim` the x -value of the right end of the rule.

```

8214     \dim_set:Nn \l_tmpb_dim
8215     { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
8216     \str_case:nn { #2 }
8217     {
8218         ) { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
8219         ] { \dim_add:Nn \l_tmpb_dim { 0.2 mm } }
8220         \} { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
8221     }
8222     \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
8223     \pgfusepathqstroke
8224     \group_end:
8225 }
8226 { \@@_error:nnn { Wrong~line~in~SubMatrix } { horizontal } { ##1 } }
8227 }

```

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).

```

8228     \str_if_empty:NF \l_@@_submatrix_name_str
8229     {
8230         \@@_pgf_rect_node:nnnnn \l_@@_submatrix_name_str
8231         \l_@@_x_initial_dim \l_@@_y_initial_dim
8232         \l_@@_x_final_dim \l_@@_y_final_dim
8233     }
8234     \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`.

```

8235     \begin { pgfscope }
8236     \pgftransformshift
8237     {
8238         \pgfpoint
8239         { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
8240         { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
8241     }
8242     \str_if_empty:NTF \l_@@_submatrix_name_str
8243     { \@@_node_left:nn #1 { } }
8244     { \@@_node_left:nn #1 { \@@_env: - \l_@@_submatrix_name_str - left } }
8245     \end { pgfscope }

```

Now, we deal with the right delimiter.

```

8246     \pgftransformshift
8247     {
8248         \pgfpoint
8249         { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
8250         { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
8251     }
8252     \str_if_empty:NTF \l_@@_submatrix_name_str
8253     { \@@_node_right:nnnn #2 { } { #3 } { #4 } }
8254     {
8255         \@@_node_right:nnnn #2

```

```

8256         { \@@_env: - \l_@@_submatrix_name_str - right } { #3 } { #4 }
8257     }

8258     \cs_set_eq:NN \pgfpointanchor \@@_pgfpointanchor:n
8259     \flag_clear_new:n { nicematrix }
8260     \l_@@_code_tl
8261 }

```

In the key code of the command `\SubMatrix` there may be Tikz instructions. We want that, in these instructions, the i and j in specifications of nodes of the forms $i-j$, $\text{row-}i$, $\text{col-}j$ and $i-|j$ refer to the number of row and column *relative* of the current `\SubMatrix`. That's why we will patch (locally in the `\SubMatrix`) the command `\pgfpointanchor`.

```

8262 \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.

```

8263 \cs_new_protected:Npn \@@_pgfpointanchor:n #1
8264 {
8265     \use:e
8266     { \exp_not:N \@@_old_pgfpointanchor { \@@_pgfpointanchor_i:nn #1 } }
8267 }

```

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 -.

```

8268 \cs_new:Npn \@@_pgfpointanchor_i:nn #1 #2
8269 { #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.

```

8270 \tl_const:Nn \c_@@_integers_alist_tl
8271 {
8272     { 1 } { } { 2 } { } { 3 } { } { 4 } { } { 5 } { }
8273     { 6 } { } { 7 } { } { 8 } { } { 9 } { } { 10 } { }
8274     { 11 } { } { 12 } { } { 13 } { } { 14 } { } { 15 } { }
8275     { 16 } { } { 17 } { } { 18 } { } { 19 } { } { 20 } { }
8276 }

8277 \cs_new:Npn \@@_pgfpointanchor_ii:w #1-#2\q_stop
8278 {

```

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`.

```

8279     \tl_if_empty:nTF { #2 }
8280     {
8281         \str_case:nVTF { #1 } \c_@@_integers_alist_tl
8282         {
8283             \flag_raise:n { nicematrix }
8284             \int_if_even:nTF { \flag_height:n { nicematrix } }
8285             { \int_eval:n { #1 + \l_@@_first_i_tl - 1 } }
8286             { \int_eval:n { #1 + \l_@@_first_j_tl - 1 } }
8287         }
8288         { #1 }
8289     }

```

If there is an hyphen, we have to see whether we have a node of the form $i-j$, $\text{row-}i$ or $\text{col-}j$.

```

8290     { \@@_pgfpointanchor_iii:w { #1 } #2 }
8291   }

```

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`).

```

8292 \cs_new:Npn \@@_pgfpointanchor_iii:w #1 #2 -
8293 {
8294   \str_case:nnF { #1 }
8295   {
8296     { row } { row - \int_eval:n { #2 + \l_@@_first_i_tl - 1 } }
8297     { col } { col - \int_eval:n { #2 + \l_@@_first_j_tl - 1 } }
8298   }

```

Now the case of a node of the form $i-j$.

```

8299   {
8300     \int_eval:n { #1 + \l_@@_first_i_tl - 1 }
8301     - \int_eval:n { #2 + \l_@@_first_j_tl - 1 }
8302   }
8303 }

```

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`).

```

8304 \cs_new_protected:Npn \@@_node_left:nn #1 #2
8305 {
8306   \pgfnode
8307   { rectangle }
8308   { east }
8309   {
8310     \nullfont
8311     \c_math_toggle_token
8312     \@@_color:V \l_@@_delimiters_color_tl
8313     \left #1
8314     \vcenter
8315     {
8316       \nullfont
8317       \hrule \@height \l_tmpa_dim
8318               \@depth \c_zero_dim
8319               \@width \c_zero_dim
8320     }
8321     \right .
8322     \c_math_toggle_token
8323   }
8324   { #2 }
8325   { }
8326 }

```

The command `\@@_node_right:nn` puts the right delimiter with the correct size. The argument `#1` is the delimiter to put. The argument `#2` is the name we will give to this PGF node (if the key `name` has been used in `\SubMatrix`). The argument `#3` is the subscript and `#4` is the superscript.

```

8327 \cs_new_protected:Npn \@@_node_right:nnnn #1 #2 #3 #4
8328 {
8329   \pgfnode
8330   { rectangle }
8331   { west }
8332   {
8333     \nullfont
8334     \c_math_toggle_token
8335     \@@_color:V \l_@@_delimiters_color_tl
8336     \left .
8337     \vcenter

```

```

8338     {
8339         \nullfont
8340         \hrule \@height \l_tmpa_dim
8341             \@depth \c_zero_dim
8342             \@width \c_zero_dim
8343     }
8344     \right #1
8345     \tl_if_empty:nF { #3 } { _ { \smash { #3 } } }
8346     ~ { \smash { #4 } }
8347     \c_math_toggle_token
8348 }
8349 { #2 }
8350 { }
8351 }

```

35 Les commandes `\UnderBrace` et `\OverBrace`

The following commands will be linked to `\UnderBrace` and `\OverBrace` in the `\CodeAfter`.

```

8352 \NewDocumentCommand \@@_UnderBrace { 0 { } m m m 0 { } }
8353 {
8354     \peek_remove_spaces:n
8355     { \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { under } }
8356 }
8357 \NewDocumentCommand \@@_OverBrace { 0 { } m m m 0 { } }
8358 {
8359     \peek_remove_spaces:n
8360     { \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { over } }
8361 }
8362 \keys_define:nn { NiceMatrix / Brace }
8363 {
8364     left-shorten .bool_set:N = \l_@@_brace_left_shorten_bool ,
8365     left-shorten .default:n = true ,
8366     right-shorten .bool_set:N = \l_@@_brace_right_shorten_bool ,
8367     shorten .meta:n = { left-shorten , right-shorten } ,
8368     right-shorten .default:n = true ,
8369     yshift .dim_set:N = \l_@@_brace_yshift_dim ,
8370     yshift .value_required:n = true ,
8371     yshift .initial:n = \c_zero_dim ,
8372     color .tl_set:N = \l_tmpa_tl ,
8373     color .value_required:n = true ,
8374     unknown .code:n = \@@_error:n { Unknown~key~for~Brace }
8375 }

```

`#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`.

```

8376 \cs_new_protected:Npn \@@_brace:nnnnn #1 #2 #3 #4 #5
8377 {
8378     \group_begin:

```

The four following token lists correspond to the position of the sub-matrix to which a brace will be attached.

```

8379 \@@_compute_i_j:nn { #1 } { #2 }
8380 \bool_lazy_or:nnTF
8381 { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
8382 { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
8383 {
8384     \str_if_eq:nnTF { #5 } { under }

```

```

8385     { \@@_error:nn { Construct~too~large } { \UnderBrace } }
8386     { \@@_error:nn { Construct~too~large } { \OverBrace } }
8387 }
8388 {
8389   \tl_clear:N \l_tmpa_tl
8390   \keys_set:nn { NiceMatrix / Brace } { #4 }
8391   \tl_if_empty:NF \l_tmpa_tl { \color { \l_tmpa_tl } }
8392   \pgfpicture
8393   \pgfrememberpicturepositiononpagetrue
8394   \pgf@relevantforpicturesizefalse
8395   \bool_if:NT \l_@@_brace_left_shorten_bool
8396   {
8397     \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
8398     \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
8399     {
8400       \cs_if_exist:cT
8401       { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
8402       {
8403         \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
8404         \dim_set:Nn \l_@@_x_initial_dim
8405         { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
8406       }
8407     }
8408   }
8409   \bool_lazy_or:nnT
8410   { \bool_not_p:n \l_@@_brace_left_shorten_bool }
8411   { \dim_compare_p:nNn \l_@@_x_initial_dim = \c_max_dim }
8412   {
8413     \@@_qpoint:n { col - \l_@@_first_j_tl }
8414     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
8415   }
8416   \bool_if:NT \l_@@_brace_right_shorten_bool
8417   {
8418     \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
8419     \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
8420     {
8421       \cs_if_exist:cT
8422       { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
8423       {
8424         \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
8425         \dim_set:Nn \l_@@_x_final_dim
8426         { \dim_max:nn \l_@@_x_final_dim \pgf@x }
8427       }
8428     }
8429   }
8430   \bool_lazy_or:nnT
8431   { \bool_not_p:n \l_@@_brace_right_shorten_bool }
8432   { \dim_compare_p:nNn \l_@@_x_final_dim = { - \c_max_dim } }
8433   {
8434     \@@_qpoint:n { col - \int_eval:n { \l_@@_last_j_tl + 1 } }
8435     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
8436   }
8437   \pgfset { inner-sep = \c_zero_dim }
8438   \str_if_eq:nnTF { #5 } { under }
8439   { \@@_underbrace_i:n { #3 } }
8440   { \@@_overbrace_i:n { #3 } }
8441   \endpgfpicture
8442 }
8443 \group_end:
8444 }

```

The argument is the text to put above the brace.

```

8445 \cs_new_protected:Npn \@@_overbrace_i:n #1
8446 {

```

```

8447 \@@_qpoint:n { row - \l_@@_first_i_tl }
8448 \pgftransformshift
8449 {
8450   \pgfpoint
8451   { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
8452   { \pgf@y + \l_@@_brace_yshift_dim - 3 pt }
8453 }
8454 \pgfnode
8455 { rectangle }
8456 { south }
8457 {
8458   \vbox_top:n
8459   {
8460     \group_begin:
8461     \everycr { }
8462     \halign
8463     {
8464       \hfil ## \hfil \crr
8465       \@@_math_toggle_token: #1 \@@_math_toggle_token: \cr
8466       \noalign { \skip_vertical:n { 3 pt } \nointerlineskip }
8467       \c_math_toggle_token
8468       \overbrace
8469       {
8470         \hbox_to_wd:nn
8471         { \l_@@_x_final_dim - \l_@@_x_initial_dim }
8472         { }
8473       }
8474       \c_math_toggle_token
8475       \cr
8476     }
8477     \group_end:
8478   }
8479 }
8480 { }
8481 { }
8482 }

```

The argument is the text to put under the brace.

```

8483 \cs_new_protected:Npn \@@_underbrace_i:n #1
8484 {
8485   \@@_qpoint:n { row - \int_eval:n { \l_@@_last_i_tl + 1 } }
8486   \pgftransformshift
8487   {
8488     \pgfpoint
8489     { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
8490     { \pgf@y - \l_@@_brace_yshift_dim + 3 pt }
8491   }
8492   \pgfnode
8493   { rectangle }
8494   { north }
8495   {
8496     \group_begin:
8497     \everycr { }
8498     \vbox:n
8499     {
8500       \halign
8501       {
8502         \hfil ## \hfil \crr
8503         \c_math_toggle_token
8504         \underbrace
8505         {
8506           \hbox_to_wd:nn
8507           { \l_@@_x_final_dim - \l_@@_x_initial_dim }
8508           { }

```

```

8509         }
8510         \c_math_toggle_token
8511         \cr
8512         \noalign { \skip_vertical:n { 3 pt } \nointerlineskip }
8513         \@@_math_toggle_token: #1 \@@_math_toggle_token: \cr
8514     }
8515 }
8516 \group_end:
8517 }
8518 { }
8519 { }
8520 }

```

36 The command \ShowCellNames

```

8521 \NewDocumentCommand \@@_ShowCellNames_CodeBefore { }
8522 {
8523     \dim_zero_new:N \g_@@_tmpc_dim
8524     \dim_zero_new:N \g_@@_tmpd_dim
8525     \dim_zero_new:N \g_@@_tmpe_dim
8526     \int_step_inline:nn \c@iRow
8527     {
8528         \begin { pgfpicture }
8529         \@@_qpoint:n { row - ##1 }
8530         \dim_set_eq:NN \l_tmpa_dim \pgf@y
8531         \@@_qpoint:n { row - \int_eval:n { ##1 + 1 } }
8532         \dim_gset:Nn \g_tmpa_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
8533         \dim_gset:Nn \g_tmpe_dim { \l_tmpa_dim - \pgf@y }
8534         \bool_if:NTF \l_@@_in_code_after_bool
8535         \end { pgfpicture }
8536         \int_step_inline:nn \c@jCol
8537         {
8538             \hbox_set:Nn \l_tmpa_box
8539             { \normalfont \Large \color { red ! 50 } ##1 - #####1 }
8540             \begin { pgfpicture }
8541             \@@_qpoint:n { col - #####1 }
8542             \dim_gset_eq:NN \g_@@_tmpc_dim \pgf@x
8543             \@@_qpoint:n { col - \int_eval:n { #####1 + 1 } }
8544             \dim_gset:Nn \g_@@_tmpd_dim { \pgf@x - \g_@@_tmpc_dim }
8545             \dim_gset_eq:NN \g_@@_tmpe_dim \pgf@x
8546             \endpgfpicture
8547             \end { pgfpicture }
8548             \fp_set:Nn \l_tmpa_fp
8549             {
8550                 \fp_min:nn
8551                 {
8552                     \fp_min:nn
8553                     { \dim_ratio:nn { \g_@@_tmpd_dim } { \box_wd:N \l_tmpa_box } }
8554                     { \dim_ratio:nn { \g_tmpe_dim } { \box_ht_plus_dp:N \l_tmpa_box } }
8555                 }
8556                 { 1.0 }
8557             }
8558             \box_scale:Nnn \l_tmpa_box { \fp_use:N \l_tmpa_fp } { \fp_use:N \l_tmpa_fp }
8559             \pgfpicture
8560             \pgfrememberpicturepositiononpagetrue
8561             \pgf@relevantforpicturesizefalse
8562             \pgftransformshift
8563             {
8564                 \pgfpoint

```

```

8565         { 0.5 * ( \g_@@_tmpc_dim + \g_@@_tmpe_dim ) }
8566         { \dim_use:N \g_tmpa_dim }
8567     }
8568     \pgfnode
8569     { rectangle }
8570     { center }
8571     { \box_use:N \l_tmpa_box }
8572     { }
8573     { }
8574     \endpgfpicture
8575 }
8576 }
8577 }
8578 \NewDocumentCommand \@@_ShowCellNames { }
8579 {
8580     \bool_if:NT \l_@@_in_code_after_bool
8581     {
8582         \pgfpicture
8583         \pgfrememberpicturepositiononpagetrue
8584         \pgf@relevantforpicturesizefalse
8585         \pgfpathrectanglecorners
8586         { \@@_qpoint:n { 1 } }
8587         {
8588             \@@_qpoint:n
8589             { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
8590         }
8591         \pgfsetfillopacity { 0.75 }
8592         \pgfsetfillcolor { white }
8593         \pgfusepathqfill
8594         \endpgfpicture
8595     }
8596     \dim_zero_new:N \g_@@_tmpc_dim
8597     \dim_zero_new:N \g_@@_tmpd_dim
8598     \dim_zero_new:N \g_@@_tmpe_dim
8599     \int_step_inline:nn \c@iRow
8600     {
8601         \bool_if:NTF \l_@@_in_code_after_bool
8602         {
8603             \pgfpicture
8604             \pgfrememberpicturepositiononpagetrue
8605             \pgf@relevantforpicturesizefalse
8606         }
8607         { \begin { pgfpicture } }
8608         \@@_qpoint:n { row - ##1 }
8609         \dim_set_eq:NN \l_tmpa_dim \pgf@y
8610         \@@_qpoint:n { row - \int_eval:n { ##1 + 1 } }
8611         \dim_gset:Nn \g_tmpa_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
8612         \dim_gset:Nn \g_tmpb_dim { \l_tmpa_dim - \pgf@y }
8613         \bool_if:NTF \l_@@_in_code_after_bool
8614         { \endpgfpicture }
8615         { \end { pgfpicture } }
8616         \int_step_inline:nn \c@jCol
8617         {
8618             \hbox_set:Nn \l_tmpa_box
8619             {
8620                 \normalfont \Large \sfseries \bfseries
8621                 \bool_if:NTF \l_@@_in_code_after_bool
8622                 { \color { red } }
8623                 { \color { red ! 50 } }
8624                 ##1 - ###1
8625             }
8626         \bool_if:NTF \l_@@_in_code_after_bool
8627         {

```

```

8628         \pgfpicture
8629         \pgfrememberpicturepositiononpagetrue
8630         \pgf@relevantforpicturesizefalse
8631     }
8632     { \begin { pgfpicture } }
8633     \@@_qpoint:n { col - #####1 }
8634     \dim_gset_eq:NN \g_@@_tmpc_dim \pgf@x
8635     \@@_qpoint:n { col - \int_eval:n { #####1 + 1 } }
8636     \dim_gset:Nn \g_@@_tmpd_dim { \pgf@x - \g_@@_tmpc_dim }
8637     \dim_gset_eq:NN \g_@@_tmpe_dim \pgf@x
8638     \bool_if:NTF \l_@@_in_code_after_bool
8639         { \endpgfpicture }
8640         { \end { pgfpicture } }
8641     \fp_set:Nn \l_tmpa_fp
8642     {
8643         \fp_min:nn
8644         {
8645             \fp_min:nn
8646             { \dim_ratio:nn { \g_@@_tmpd_dim } { \box_wd:N \l_tmpa_box } }
8647             { \dim_ratio:nn { \g_tmpb_dim } { \box_ht_plus_dp:N \l_tmpa_box } }
8648         }
8649         { 1.0 }
8650     }
8651     \box_scale:Nnn \l_tmpa_box { \fp_use:N \l_tmpa_fp } { \fp_use:N \l_tmpa_fp }
8652     \pgfpicture
8653     \pgfrememberpicturepositiononpagetrue
8654     \pgf@relevantforpicturesizefalse
8655     \pgftransformshift
8656     {
8657         \pgfpoint
8658         { 0.5 * ( \g_@@_tmpc_dim + \g_@@_tmpe_dim ) }
8659         { \dim_use:N \g_tmpa_dim }
8660     }
8661     \pgfnode
8662     { rectangle }
8663     { center }
8664     { \box_use:N \l_tmpa_box }
8665     { }
8666     { }
8667     \endpgfpicture
8668 }
8669 }
8670 }

```

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.

```
8671 \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.

```

8672 \bool_new:N \g_@@_footnote_bool
8673 \msg_new:nnnn { nicematrix } { Unknown-key-for~package }
8674 {
8675     The~key~'\l_keys_key_str'~is~unknown. \\

```

```

8676   That~key~will~be~ignored. \\
8677   For~a~list~of~the~available~keys,~type~H~<return>.
8678 }
8679 {
8680   The~available~keys~are~(in~alphabetic~order):~
8681   footnote,~
8682   footnotehyper,~
8683   messages-for-Overleaf,~
8684   no-test-for-array,~
8685   renew-dots,~and~
8686   renew-matrix.
8687 }
8688 \keys_define:nn { NiceMatrix / Package }
8689 {
8690   renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
8691   renew-dots .value_forbidden:n = true ,
8692   renew-matrix .code:n = \@@_renew_matrix: ,
8693   renew-matrix .value_forbidden:n = true ,
8694   messages-for-Overleaf .bool_set:N = \g_@@_messages_for_Overleaf_bool ,
8695   footnote .bool_set:N = \g_@@_footnote_bool ,
8696   footnotehyper .bool_set:N = \g_@@_footnotehyper_bool ,
8697   no-test-for-array .bool_set:N = \g_@@_no_test_for_array_bool ,
8698   no-test-for-array .default:n = true ,
8699   unknown .code:n = \@@_error:n { Unknown~key~for~package }
8700 }
8701 \ProcessKeysOptions { NiceMatrix / Package }

8702 \@@_msg_new:nn { footnote~with~footnotehyper~package }
8703 {
8704   You~can't~use~the~option~'footnote'~because~the~package~
8705   footnotehyper~has~already~been~loaded.~
8706   If~you~want,~you~can~use~the~option~'footnotehyper'~and~the~footnotes~
8707   within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
8708   of~the~package~footnotehyper.\\
8709   The~package~footnote~won't~be~loaded.
8710 }
8711 \@@_msg_new:nn { footnotehyper~with~footnote~package }
8712 {
8713   You~can't~use~the~option~'footnotehyper'~because~the~package~
8714   footnote~has~already~been~loaded.~
8715   If~you~want,~you~can~use~the~option~'footnote'~and~the~footnotes~
8716   within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
8717   of~the~package~footnote.\\
8718   The~package~footnotehyper~won't~be~loaded.
8719 }

8720 \bool_if:NT \g_@@_footnote_bool
8721 {
The class beamer has its own system to extract footnotes and that's why we have nothing to do if
beamer is used.
8722   \IfClassLoadedTF { beamer }
8723   { \bool_set_false:N \g_@@_footnote_bool }
8724   {
8725     \IfPackageLoadedTF { footnotehyper }
8726     { \@@_error:n { footnote~with~footnotehyper~package } }
8727     { \usepackage { footnote } }
8728   }
8729 }
8730 \bool_if:NT \g_@@_footnotehyper_bool
8731 {

```

The class `beamer` has its own system to extract footnotes and that's why we have nothing to do if `beamer` is used.

```

8732 \IfClassLoadedTF { beamer }
8733 { \bool_set_false:N \g_@@_footnote_bool }
8734 {
8735   \IfPackageLoadedTF { footnote }
8736   { \@@_error:n { footnotehyper~with~footnote-package } }
8737   { \usepackage { footnotehyper } }
8738 }
8739 \bool_set_true:N \g_@@_footnote_bool
8740 }

```

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.

```

8741 \bool_new:N \l_@@_underscore_loaded_bool
8742 \IfPackageLoadedTF { underscore }
8743 { \bool_set_true:N \l_@@_underscore_loaded_bool }
8744 { }
8745 \hook_gput_code:nnn { begindocument } { . }
8746 {
8747   \bool_if:NF \l_@@_underscore_loaded_bool
8748   {
8749     \IfPackageLoadedTF { underscore }
8750     { \@@_error:n { underscore~after~nicematrix } }
8751     { }
8752   }
8753 }

```

39 Error messages of the package

```

8754 \bool_if:NTF \g_@@_messages_for_Overleaf_bool
8755 { \str_const:Nn \c_@@_available_keys_str { } }
8756 {
8757   \str_const:Nn \c_@@_available_keys_str
8758   { For~a~list~of~the~available~keys,~type~H~<return>. }
8759 }
8760 \seq_new:N \g_@@_types_of_matrix_seq
8761 \seq_gset_from_clist:Nn \g_@@_types_of_matrix_seq
8762 {
8763   NiceMatrix ,
8764   pNiceMatrix , bNiceMatrix , vNiceMatrix, BNiceMatrix, VNiceMatrix
8765 }
8766 \seq_gset_map_x:NNn \g_@@_types_of_matrix_seq \g_@@_types_of_matrix_seq
8767 { \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`.

```

8768 \cs_new_protected:Npn \@@_error_too_much_cols:
8769 {
8770   \seq_if_in:NVTf \g_@@_types_of_matrix_seq \g_@@_name_env_str
8771   {
8772     \int_compare:nNtF \l_@@_last_col_int = { -2 }
8773     { \@@_fatal:n { too-much-cols-for-matrix } }
8774     {
8775       \int_compare:nNtF \l_@@_last_col_int = { -1 }
8776       { \@@_fatal:n { too-much-cols-for-matrix } }
8777       {
8778         \bool_if:NF \l_@@_last_col_without_value_bool
8779         { \@@_fatal:n { too-much-cols-for-matrix-with-last-col } }
8780       }
8781     }
8782   }
8783   {
8784     \IfPackageLoadedTF { tabularx }
8785     {
8786       \str_if_eq:VnTF \g_@@_name_env_str { NiceTabularX }
8787       {
8788         \int_compare:nNtF \c@iRow = \c_zero_int
8789         { \@@_fatal:n { X-columns-with-tabularx } }
8790         {
8791           \@@_fatal:nn { too-much-cols-for-array }
8792           {
8793             However,~this-message-may-be-erroneous:~
8794             maybe-you-have-used-X-columns-while-'tabularx'-is-loaded,~
8795             ~which-is-forbidden-(however,~it's-still-possible-to-use~
8796             X-columns-in~{NiceTabularX}).
8797           }
8798         }
8799       }
8800       { \@@_fatal:nn { too-much-cols-for-array } { } }
8801     }
8802     { \@@_fatal:nn { too-much-cols-for-array } { } }
8803   }
8804 }

```

The following command must *not* be protected since it's used in an error message.

```

8805 \cs_new:Npn \@@_message_hdotsfor:
8806 {
8807   \tl_if_empty:VF \g_@@_HVdotsfor_lines_tl
8808   { ~Maybe-your-use-of~\token_to_str:N \Hdotsfor\ is-incorrect.}
8809 }
8810 \@@_msg_new:nn { hvlines,~rounded-corners-and~corners }
8811 {
8812   Incompatible~options.\\
8813   You-should-not-use-'hvlines',~'rounded-corners'-and-'corners'-at~this~time.\\
8814   The~output-will-not-be-reliable.
8815 }
8816 \@@_msg_new:nn { negative-weight }
8817 {
8818   Negative-weight.\\
8819   The~weight-of~the-'X'~columns-must-be-positive-and-you-have-used~
8820   the-value~'\int_use:N \l_@@_weight_int'.\\
8821   The~absolute-value-will-be-used.
8822 }
8823 \@@_msg_new:nn { last-col-not-used }
8824 {
8825   Column-not-used.\\
8826   The-key~'last-col'~is-in-force-but-you-have-not-used-that~last~column~

```

```

8827     in~your~\@@_full_name_env:~However,~you~can~go~on.
8828 }

8829 \@@_msg_new:nn { too~much~cols~for~matrix~with~last~col }
8830 {
8831     Too~much~columns.\\
8832     In~the~row~\int_eval:n { \c@iRow },~
8833     you~try~to~use~more~columns~
8834     than~allowed~by~your~\@@_full_name_env:~\@@_message_hdotsfor:\
8835     The~maximal~number~of~columns~is~\int_eval:n { \l_@@_last_col_int - 1 }~
8836     (plus~the~exterior~columns).~This~error~is~fatal.
8837 }

8838 \@@_msg_new:nn { too~much~cols~for~matrix }
8839 {
8840     Too~much~columns.\\
8841     In~the~row~\int_eval:n { \c@iRow },~
8842     you~try~to~use~more~columns~than~allowed~by~your~
8843     \@@_full_name_env:~\@@_message_hdotsfor:\ Recall~that~the~maximal~
8844     number~of~columns~for~a~matrix~(excepted~the~potential~exterior~
8845     columns)~is~fixed~by~the~LaTeX~counter~'MaxMatrixCols'.~
8846     Its~current~value~is~\int_use:N \c@MaxMatrixCols\ (use~
8847     \token_to_str:N \setcounter\ to~change~that~value).~
8848     This~error~is~fatal.
8849 }

8850 \@@_msg_new:nn { too~much~cols~for~array }
8851 {
8852     Too~much~columns.\\
8853     In~the~row~\int_eval:n { \c@iRow },~
8854     ~you~try~to~use~more~columns~than~allowed~by~your~
8855     \@@_full_name_env:~\@@_message_hdotsfor:\ The~maximal~number~of~columns~is~
8856     \int_use:N \g_@@_static_num_of_col_int\
8857     ~(plus~the~potential~exterior~ones).~#1
8858     This~error~is~fatal.
8859 }

8860 \@@_msg_new:nn { X~columns~with~tabularx }
8861 {
8862     There~is~a~problem.\\
8863     You~have~probably~used~X~columns~in~your~environment~{\g_@@_name_env_str}.~
8864     That's~not~allowed~because~'tabularx'~is~loaded~(however,~you~can~use~X~columns~
8865     in~an~environment~{NiceTabularX}).\\
8866     This~error~is~fatal.
8867 }

8868 \@@_msg_new:nn { columns~not~used }
8869 {
8870     Columns~not~used.\\
8871     The~preamble~of~your~\@@_full_name_env:\ announces~\int_use:N
8872     \g_@@_static_num_of_col_int\ columns~but~you~use~only~\int_use:N \c@jCol.\\
8873     The~columns~you~did~not~used~won't~be~created.\\
8874     You~won't~have~similar~error~till~the~end~of~the~document.
8875 }

8876 \@@_msg_new:nn { in~first~col }
8877 {
8878     Erroneous~use.\\
8879     You~can't~use~the~command~#1 in~the~first~column~(number~0)~of~the~array.\\
8880     That~command~will~be~ignored.
8881 }

8882 \@@_msg_new:nn { in~last~col }
8883 {
8884     Erroneous~use.\\
8885     You~can't~use~the~command~#1 in~the~last~column~(exterior)~of~the~array.\\
8886     That~command~will~be~ignored.

```

```

8887 }
8888 \@@_msg_new:nn { in-first-row }
8889 {
8890   Erroneous-use.\
8891   You-can't-use-the-command~#1 in-the-first-row~(number~0)-of-the-array.\
8892   That-command-will-be-ignored.
8893 }
8894 \@@_msg_new:nn { in-last-row }
8895 {
8896   You-can't-use-the-command~#1 in-the-last-row~(exterior)-of-the-array.\
8897   That-command-will-be-ignored.
8898 }
8899 \@@_msg_new:nn { caption-outside-float }
8900 {
8901   Key-caption-forbidden.\
8902   You-can't-use-the-key~'caption'~because-you-are-not-in-a-floating~
8903   environment.~This-key-will-be-ignored.
8904 }
8905 \@@_msg_new:nn { short-caption-without-caption }
8906 {
8907   You-should-not-use-the-key~'short-caption'~without~'caption'.~
8908   However,~your~'short-caption'~will-be-used-as~'caption'.
8909 }
8910 \@@_msg_new:nn { double-closing-delimiter }
8911 {
8912   Double-delimiter.\
8913   You-can't-put-a-second-closing-delimiter~"#1"~just-after-a-first-closing~
8914   delimiter.~This-delimiter-will-be-ignored.
8915 }
8916 \@@_msg_new:nn { delimiter-after-opening }
8917 {
8918   Double-delimiter.\
8919   You-can't-put-a-second-delimiter~"#1"~just-after-a-first-opening~
8920   delimiter.~That-delimiter-will-be-ignored.
8921 }
8922 \@@_msg_new:nn { bad-option-for-line-style }
8923 {
8924   Bad-line-style.\
8925   Since-you-haven't-loaded-Tikz,~the-only-value-you-can-give-to~'line-style'~
8926   is~'standard'.~That-key-will-be-ignored.
8927 }
8928 \@@_msg_new:nn { Identical-notes-in-caption }
8929 {
8930   Identical-tabular-notes.\
8931   You-can't-put-several-notes-with-the-same-content-in~
8932   \token_to_str:N \caption\ (but-you-can-in-the-main-tabular).\
8933   If-you-go-on,~the-output-will-probably-be-erroneous.
8934 }
8935 \@@_msg_new:nn { tabularnote-below-the-tabular }
8936 {
8937   \token_to_str:N \tabularnote\ forbidden\
8938   You-can't-use~\token_to_str:N \tabularnote\ in-the-caption~
8939   of-your-tabular~because-the-caption-will-be-composed-below~
8940   the-tabular.~If-you-want-the-caption-above-the-tabular~use-the~
8941   key~'caption-above'~in~\token_to_str:N \NiceMatrixOptions.\
8942   Your~\token_to_str:N \tabularnote\ will-be-discarded-and~
8943   no-similar-error-will-be-raised-in-this-document.
8944 }
8945 \@@_msg_new:nn { Unknown-key-for-rules }

```

```

8946 {
8947   Unknown~key.\\
8948   There-is-only-two-keys-available-here:~width-and-color.\\
8949   Your-key~'\l_keys_key_str'~will-be-ignored.
8950 }
8951 \@@_msg_new:nn { Unknown~key-for~rotate }
8952 {
8953   Unknown~key.\\
8954   The~only~key~available~here~is~'c'.\\
8955   Your-key~'\l_keys_key_str'~will-be-ignored.
8956 }
8957 \@@_msg_new:nnn { Unknown~key-for~custom-line }
8958 {
8959   Unknown~key.\\
8960   The~key~'\l_keys_key_str'~is-unknown~in~a~'custom-line'.~
8961   It~you-go-on,~you-will-probably-have-other-errors. \\
8962   \c_@@_available_keys_str
8963 }
8964 {
8965   The~available~keys~are~(in~alphabetic~order):~
8966   ccommand,~
8967   color,~
8968   command,~
8969   dotted,~
8970   letter,~
8971   multiplicity,~
8972   sep-color,~
8973   tikz,~and~total-width.
8974 }
8975 \@@_msg_new:nnn { Unknown~key-for~xdots }
8976 {
8977   Unknown~key.\\
8978   The~key~'\l_keys_key_str'~is-unknown~for~a~command~for~drawing~dotted~rules.\\
8979   \c_@@_available_keys_str
8980 }
8981 {
8982   The~available~keys~are~(in~alphabetic~order):~
8983   'color',~
8984   'horizontal-labels',~
8985   'inter',~
8986   'line-style',~
8987   'radius',~
8988   'shorten',~
8989   'shorten-end'~and~'shorten-start'.
8990 }
8991 \@@_msg_new:nn { Unknown~key-for~rowcolors }
8992 {
8993   Unknown~key.\\
8994   As~for~now,~there-is-only-two-keys-available-here:~'cols'~and~'respect-blocks'~
8995   (and~you~try~to~use~'\l_keys_key_str')\\
8996   That~key~will~be~ignored.
8997 }
8998 \@@_msg_new:nn { label~without~caption }
8999 {
9000   You~can't~use~the~key~'label'~in~your~'{NiceTabular}'~because~
9001   you~have~not~used~the~key~'caption'.~The~key~'label'~will~be~ignored.
9002 }
9003 \@@_msg_new:nn { W~warning }
9004 {
9005   Line~\msg_line_number:~The~cell~is~too~wide~for~your~column~'W'~
9006   (row~\int_use:N \c@iRow).

```

```

9007     }
9008 \@@_msg_new:nn { Construct~too~large }
9009 {
9010     Construct~too~large.\\
9011     Your~command~\token_to_str:N #1
9012     can't~be~drawn~because~your~matrix~is~too~small.\\
9013     That~command~will~be~ignored.
9014 }
9015 \@@_msg_new:nn { underscore~after~nicematrix }
9016 {
9017     Problem~with~'underscore'.\\
9018     The~package~'underscore'~should~be~loaded~before~'nicematrix'.~
9019     You~can~go~on~but~you~won't~be~able~to~write~something~such~as:\\
9020     '\token_to_str:N \Cdots\token_to_str:N _{n~\token_to_str:N \text{~times}}'.
9021 }
9022 \@@_msg_new:nn { ampersand~in~light-syntax }
9023 {
9024     Ampersand~forbidden.\\
9025     You~can't~use~an~ampersand~(\token_to_str:N &)~to~separate~columns~because~
9026     ~the~key~'light-syntax'~is~in~force.~This~error~is~fatal.
9027 }
9028 \@@_msg_new:nn { double~backslash~in~light-syntax }
9029 {
9030     Double~backslash~forbidden.\\
9031     You~can't~use~\token_to_str:N
9032     \\~to~separate~rows~because~the~key~'light-syntax'~
9033     is~in~force.~You~must~use~the~character~'\l_@@_end_of_row_tl'~
9034     (set~by~the~key~'end-of-row').~This~error~is~fatal.
9035 }
9036 \@@_msg_new:nn { hlines~with~color }
9037 {
9038     Incompatible~keys.\\
9039     You~can't~use~the~keys~'hlines',~'vlines'~or~'hvlines'~for~a~
9040     '\token_to_str:N \Block'~when~the~key~'color'~or~'draw'~is~used.\\
9041     Maybe~it~will~possible~in~future~version.\\
9042     Your~key~will~be~discarded.
9043 }
9044 \@@_msg_new:nn { bad~value~for~baseline }
9045 {
9046     Bad~value~for~baseline.\\
9047     The~value~given~to~'baseline'~(\int_use:N \l_tmpa_int)~is~not~
9048     valid.~The~value~must~be~between~\int_use:N \l_@@_first_row_int\ and~
9049     \int_use:N \g_@@_row_total_int\ or~equal~to~'t',~'c'~or~'b'~or~of~
9050     the~form~'line-i'.\\
9051     A~value~of~1~will~be~used.
9052 }
9053 \@@_msg_new:nn { ragged2e~not~loaded }
9054 {
9055     You~have~to~load~'ragged2e'~in~order~to~use~the~key~'\l_keys_key_str'~in~
9056     your~column~'\l_@@_vpos_col_str'~(or~'X').~The~key~'\str_lowercase:V
9057     \l_keys_key_str'~will~be~used~instead.
9058 }
9059 \@@_msg_new:nn { Invalid~name }
9060 {
9061     Invalid~name.\\
9062     You~can't~give~the~name~'\l_keys_value_tl'~to~a~\token_to_str:N
9063     \SubMatrix\ of~your~\@@_full_name_env:.\\
9064     A~name~must~be~accepted~by~the~regular~expression~[A-Za-z][A-Za-z0-9]*.\\
9065     This~key~will~be~ignored.
9066 }

```

```

9067 \@@_msg_new:nn { Wrong-line-in-SubMatrix }
9068 {
9069   Wrong-line.\\
9070   You~try~to~draw~a~#1~line~of~number~'#2'~in~a~
9071   \token_to_str:N \SubMatrix\ of~your~\@@_full_name_env:\ but~that~
9072   number~is~not~valid.~It~will~be~ignored.
9073 }
9074 \@@_msg_new:nn { Impossible-delimiter }
9075 {
9076   Impossible-delimiter.\\
9077   It's~impossible~to~draw~the~#1~delimiter~of~your~
9078   \token_to_str:N \SubMatrix\ because~all~the~cells~are~empty~
9079   in~that~column.
9080   \bool_if:NT \l_@@_submatrix_slim_bool
9081     { ~Maybe~you~should~try~without~the~key~'slim'. } \\
9082   This~\token_to_str:N \SubMatrix\ will~be~ignored.
9083 }
9084 \@@_msg_new:nnn { width-without-X-columns }
9085 {
9086   You~have~used~the~key~'width'~but~you~have~put~no~'X'~column.~
9087   That~key~will~be~ignored.
9088 }
9089 {
9090   This~message~is~the~message~'width~without~X~columns'~
9091   of~the~module~'nicematrix'.~
9092   The~experimented~users~can~disable~that~message~with~
9093   \token_to_str:N \msg_redirect_name:nnn.\\
9094 }
9095
9096 \@@_msg_new:nn { key-multiplicity-with-dotted }
9097 {
9098   Incompatible~keys. \\
9099   You~have~used~the~key~'multiplicity'~with~the~key~'dotted'~
9100   in~a~'custom-line'.~They~are~incompatible. \\
9101   The~key~'multiplicity'~will~be~discarded.
9102 }
9103 \@@_msg_new:nn { empty-environment }
9104 {
9105   Empty~environment.\\
9106   Your~\@@_full_name_env:\ is~empty.~This~error~is~fatal.
9107 }
9108 \@@_msg_new:nn { No-letter-and-no-command }
9109 {
9110   Erroneous~use.\\
9111   Your~use~of~'custom-line'~is~no~op~since~you~don't~have~used~the~
9112   key~'letter'~(for~a~letter~for~vertical~rules)~nor~the~keys~'command'~or~
9113   '~'ccommand'~(to~draw~horizontal~rules).\\
9114   However,~you~can~go~on.
9115 }
9116 \@@_msg_new:nn { Forbidden-letter }
9117 {
9118   Forbidden~letter.\\
9119   You~can't~use~the~letter~'\l_@@_letter_str'~for~a~customized~line.\\
9120   It~will~be~ignored.
9121 }
9122 \@@_msg_new:nn { Several-letters }
9123 {
9124   Wrong~name.\\
9125   You~must~use~only~one~letter~as~value~for~the~key~'letter'~(and~you~
9126   have~used~'\l_@@_letter_str').\\
9127   It~will~be~ignored.

```

```

9128 }
9129 \@@_msg_new:nn { Delimiter~with~small }
9130 {
9131   Delimiter~forbidden.\
9132   You~can't~put~a~delimiter~in~the~preamble~of~your~\@@_full_name_env:\
9133   because~the~key~'small'~is~in~force.\
9134   This~error~is~fatal.
9135 }
9136 \@@_msg_new:nn { unknown~cell~for~line~in~CodeAfter }
9137 {
9138   Unknown~cell.\
9139   Your~command~\token_to_str:N\line\{#1\}\{#2\}~in~
9140   the~\token_to_str:N \CodeAfter\ of~your~\@@_full_name_env:\
9141   can't~be~executed~because~a~cell~doesn't~exist.\
9142   This~command~\token_to_str:N \line\ will~be~ignored.
9143 }
9144 \@@_msg_new:nnn { Duplicate~name~for~SubMatrix }
9145 {
9146   Duplicate~name.\
9147   The~name~'#1'~is~already~used~for~a~\token_to_str:N \SubMatrix\
9148   in~this~\@@_full_name_env:.\
9149   This~key~will~be~ignored.\
9150   \bool_if:NF \g_@@_messages_for_Overleaf_bool
9151   { For~a~list~of~the~names~already~used,~type~H~<return>. }
9152 }
9153 {
9154   The~names~already~defined~in~this~\@@_full_name_env:\ are:~
9155   \seq_use:Nnnn \g_@@_submatrix_names_seq { ~and~ } { ,~ } { ~and~ }.
9156 }
9157 \@@_msg_new:nn { r~or~l~with~preamble }
9158 {
9159   Erroneous~use.\
9160   You~can't~use~the~key~'\l_keys_key_str'~in~your~\@@_full_name_env:~
9161   You~must~specify~the~alignment~of~your~columns~with~the~preamble~of~
9162   your~\@@_full_name_env:.\
9163   This~key~will~be~ignored.
9164 }
9165 \@@_msg_new:nn { Hdotsfor~in~col~0 }
9166 {
9167   Erroneous~use.\
9168   You~can't~use~\token_to_str:N \Hdotsfor\ in~an~exterior~column~of~
9169   the~array.~This~error~is~fatal.
9170 }
9171 \@@_msg_new:nn { bad~corner }
9172 {
9173   Bad~corner.\
9174   #1~is~an~incorrect~specification~for~a~corner~(in~the~key~
9175   'corners').~The~available~values~are:~NW,~SW,~NE~and~SE.\
9176   This~specification~of~corner~will~be~ignored.
9177 }
9178 \@@_msg_new:nn { bad~border }
9179 {
9180   Bad~border.\
9181   \l_keys_key_str\space~is~an~incorrect~specification~for~a~border~
9182   (in~the~key~'borders'~of~the~command~\token_to_str:N \Block).~
9183   The~available~values~are:~left,~right,~top~and~bottom~(and~you~can~
9184   also~use~the~key~'tikz'
9185   \IfPackageLoadedTF { tikz }
9186   { }
9187   {~if~you~load~the~LaTeX~package~'tikz'}).\
9188   This~specification~of~border~will~be~ignored.

```

```

9189 }
9190 \@@_msg_new:nn { tikz~key~without~tikz }
9191 {
9192   Tikz~not~loaded.\\
9193   You~can't~use~the~key~'tikz'~for~the~command~'\token_to_str:N
9194   \Block'~because~you~have~not~loaded~tikz.~
9195   This~key~will~be~ignored.
9196 }
9197 \@@_msg_new:nn { last-col~non-empty~for~NiceArray }
9198 {
9199   Erroneous~use.\\
9200   In~the~\@@_full_name_env:,~you~must~use~the~key~
9201   'last-col'~without~value.\\
9202   However,~you~can~go~on~for~this~time~
9203   (the~value~'\l_keys_value_tl'~will~be~ignored).
9204 }
9205 \@@_msg_new:nn { last-col~non-empty~for~NiceMatrixOptions }
9206 {
9207   Erroneous~use.\\
9208   In~\token_to_str:N \NiceMatrixOptions,~you~must~use~the~key~
9209   'last-col'~without~value.\\
9210   However,~you~can~go~on~for~this~time~
9211   (the~value~'\l_keys_value_tl'~will~be~ignored).
9212 }
9213 \@@_msg_new:nn { Block~too~large~1 }
9214 {
9215   Block~too~large.\\
9216   You~try~to~draw~a~block~in~the~cell~#1-#2~of~your~matrix~but~the~matrix~is~
9217   too~small~for~that~block. \\
9218 }
9219 \@@_msg_new:nn { Block~too~large~2 }
9220 {
9221   Block~too~large.\\
9222   The~preamble~of~your~\@@_full_name_env:\ announces~\int_use:N
9223   \g_@@_static_num_of_col_int\
9224   columns~but~you~use~only~\int_use:N \c@jCol\ and~that's~why~a~block~
9225   specified~in~the~cell~#1-#2~can't~be~drawn.~You~should~add~some~ampersands~
9226   (&)~at~the~end~of~the~first~row~of~your~
9227   \@@_full_name_env:.\\
9228   This~block~and~maybe~others~will~be~ignored.
9229 }
9230 \@@_msg_new:nn { unknown~column~type }
9231 {
9232   Bad~column~type.\\
9233   The~column~type~'#1'~in~your~\@@_full_name_env:\
9234   is~unknown. \\
9235   This~error~is~fatal.
9236 }
9237 \@@_msg_new:nn { unknown~column~type~S }
9238 {
9239   Bad~column~type.\\
9240   The~column~type~'S'~in~your~\@@_full_name_env:\ is~unknown. \\
9241   If~you~want~to~use~the~column~type~'S'~of~siunitx,~you~should~
9242   load~that~package. \\
9243   This~error~is~fatal.
9244 }
9245 \@@_msg_new:nn { tabularnote~forbidden }
9246 {
9247   Forbidden~command.\\
9248   You~can't~use~the~command~\token_to_str:N\tabularnote\

```

```

9249 ~here.~This~command~is~available~only~in~
9250 \{NiceTabular\},~\{NiceTabular*\}~and~\{NiceTabularX\}~or~in~
9251 the~argument~of~a~command~\token_to_str:N \caption\ included~
9252 in~an~environment~{table}. \\
9253 This~command~will~be~ignored.
9254 }
9255 \@@_msg_new:nn { borders~forbidden }
9256 {
9257   Forbidden~key.\\
9258   You~can't~use~the~key~'borders'~of~the~command~\token_to_str:N \Block\
9259   because~the~option~'rounded-corners'~
9260   is~in~force~with~a~non-zero~value.\\
9261   This~key~will~be~ignored.
9262 }
9263 \@@_msg_new:nn { bottomrule~without~booktabs }
9264 {
9265   booktabs~not~loaded.\\
9266   You~can't~use~the~key~'tabular/bottomrule'~because~you~haven't~
9267   loaded~'booktabs'.\\
9268   This~key~will~be~ignored.
9269 }
9270 \@@_msg_new:nn { enumitem~not~loaded }
9271 {
9272   enumitem~not~loaded.\\
9273   You~can't~use~the~command~\token_to_str:N\tabularnote\
9274   ~because~you~haven't~loaded~'enumitem'.\\
9275   All~the~commands~\token_to_str:N\tabularnote\ will~be~
9276   ignored~in~the~document.
9277 }
9278 \@@_msg_new:nn { tikz~in~custom~line~without~tikz }
9279 {
9280   Tikz~not~loaded.\\
9281   You~have~used~the~key~'tikz'~in~the~definition~of~a~
9282   customized~line~(with~'custom~line')~but~tikz~is~not~loaded.~
9283   You~can~go~on~but~you~will~have~another~error~if~you~actually~
9284   use~that~custom~line.
9285 }
9286 \@@_msg_new:nn { tikz~in~borders~without~tikz }
9287 {
9288   Tikz~not~loaded.\\
9289   You~have~used~the~key~'tikz'~in~a~key~'borders'~(of~a~
9290   command~\token_to_str:N\Block')~but~tikz~is~not~loaded.~
9291   That~key~will~be~ignored.
9292 }
9293 \@@_msg_new:nn { color~in~custom~line~with~tikz }
9294 {
9295   Erroneous~use.\\
9296   In~a~'custom~line',~you~have~used~both~'tikz'~and~'color',~
9297   which~is~forbidden~(you~should~use~'color'~inside~the~key~'tikz').~
9298   The~key~'color'~will~be~discarded.
9299 }
9300 \@@_msg_new:nn { Wrong~last~row }
9301 {
9302   Wrong~number.\\
9303   You~have~used~'last~row=\int_use:N \l_@@_last_row_int'~but~your~
9304   \@@_full_name_env:\ seems~to~have~\int_use:N \c@iRow \ rows.~
9305   If~you~go~on,~the~value~of~\int_use:N \c@iRow \ will~be~used~for~
9306   last~row.~You~can~avoid~this~problem~by~using~'last~row'~
9307   without~value~(more~compilations~might~be~necessary).
9308 }

```

```

9309 \@@_msg_new:nn { Yet~in~env }
9310 {
9311   Nested~environments.\\
9312   Environments~of~nicematrix~can't~be~nested.\\
9313   This~error~is~fatal.
9314 }
9315 \@@_msg_new:nn { Outside~math~mode }
9316 {
9317   Outside~math~mode.\\
9318   The~\@@_full_name_env:\ can~be~used~only~in~math~mode~
9319   (and~not~in~\token_to_str:N \vcenter).\\
9320   This~error~is~fatal.
9321 }
9322 \@@_msg_new:nn { One~letter~allowed }
9323 {
9324   Bad~name.\\
9325   The~value~of~key~'\l_keys_key_str'~must~be~of~length~1.\\
9326   It~will~be~ignored.
9327 }
9328 \@@_msg_new:nn { TabularNote~in~CodeAfter }
9329 {
9330   Environment~{TabularNote}~forbidden.\\
9331   You~must~use~{TabularNote}~at~the~end~of~your~{NiceTabular}~
9332   but~*before*~the~\token_to_str:N \CodeAfter.\\
9333   This~environment~{TabularNote}~will~be~ignored.
9334 }
9335 \@@_msg_new:nn { varwidth~not~loaded }
9336 {
9337   varwidth~not~loaded.\\
9338   You~can't~use~the~column~type~'V'~because~'varwidth'~is~not~
9339   loaded.\\
9340   Your~column~will~behave~like~'p'.
9341 }
9342 \@@_msg_new:nnn { Unknow~key~for~RulesBis }
9343 {
9344   Unknow~key.\\
9345   Your~key~'\l_keys_key_str'~is~unknown~for~a~rule.\\
9346   \c_@@_available_keys_str
9347 }
9348 {
9349   The~available~keys~are~(in~alphabetic~order):~
9350   color,~
9351   dotted,~
9352   multiplicity,~
9353   sep-color,~
9354   tikz,~and~total-width.
9355 }
9356
9357 \@@_msg_new:nnn { Unknown~key~for~Block }
9358 {
9359   Unknown~key.\\
9360   The~key~'\l_keys_key_str'~is~unknown~for~the~command~\token_to_str:N
9361   \Block.\\ It~will~be~ignored. \\
9362   \c_@@_available_keys_str
9363 }
9364 {
9365   The~available~keys~are~(in~alphabetic~order):~b,~B,~borders,~c,~draw,~fill,~
9366   hlines,~hvlines,~l,~line-width,~name,~opacity,~rounded-corners,~r,~
9367   respect-arraystretch,~t,~T,~tikz,~transparent~and~vlines.
9368 }
9369 \@@_msg_new:nn { Version~of~siunitx~too~old }

```

```

9370 {
9371     siunitx~too~old.\\
9372     You~can't~use~'S'~columns~because~your~version~of~'siunitx'~
9373     is~too~old.~You~need~at~least~v~3.0.38~and~your~log~file~says:~"siunitx,~
9374     \use:c { ver @ siunitx.sty }". \\
9375     This~error~is~fatal.
9376 }
9377 \@@_msg_new:nnn { Unknown~key~for~Brace }
9378 {
9379     Unknown~key.\\
9380     The~key~'\l_keys_key_str'~is~unknown~for~the~commands~\token_to_str:N
9381     \UnderBrace\ and~\token_to_str:N \OverBrace.\\
9382     It~will~be~ignored. \\
9383     \c_@@_available_keys_str
9384 }
9385 {
9386     The~available~keys~are~(in~alphabetic~order):~color,~left~shorten,~
9387     right~shorten,~shorten~(which~fixes~both~left~shorten~and~
9388     right~shorten)~and~yshift.
9389 }
9390 \@@_msg_new:nnn { Unknown~key~for~CodeAfter }
9391 {
9392     Unknown~key.\\
9393     The~key~'\l_keys_key_str'~is~unknown.\\
9394     It~will~be~ignored. \\
9395     \c_@@_available_keys_str
9396 }
9397 {
9398     The~available~keys~are~(in~alphabetic~order):~
9399     delimiters/color,~
9400     rules~(with~the~subkeys~'color'~and~'width'),~
9401     sub-matrix~(several~subkeys)~
9402     and~xdots~(several~subkeys).~
9403     The~latter~is~for~the~command~\token_to_str:N \line.
9404 }
9405 \@@_msg_new:nnn { Unknown~key~for~CodeBefore }
9406 {
9407     Unknown~key.\\
9408     The~key~'\l_keys_key_str'~is~unknown.\\
9409     It~will~be~ignored. \\
9410     \c_@@_available_keys_str
9411 }
9412 {
9413     The~available~keys~are~(in~alphabetic~order):~
9414     create~cell~nodes,~
9415     delimiters/color~and~
9416     sub-matrix~(several~subkeys).
9417 }
9418 \@@_msg_new:nnn { Unknown~key~for~SubMatrix }
9419 {
9420     Unknown~key.\\
9421     The~key~'\l_keys_key_str'~is~unknown.\\
9422     That~key~will~be~ignored. \\
9423     \c_@@_available_keys_str
9424 }
9425 {
9426     The~available~keys~are~(in~alphabetic~order):~
9427     'delimiters/color',~
9428     'extra-height',~
9429     'hlines',~
9430     'hvlines',~
9431     'left-xshift',~

```

```

9432     'name',~
9433     'right-xshift',~
9434     'rules'~(with~the~subkeys~'color'~and~'width'),~
9435     'slim',~
9436     'vlines'~and~'xshift'~(which~sets~both~'left-xshift'~
9437     and~'right-xshift').\\
9438 }
9439 \@@_msg_new:nnn { Unknown~key~for~notes }
9440 {
9441     Unknown~key.\\
9442     The~key~'\l_keys_key_str'~is~unknown.\\
9443     That~key~will~be~ignored. \\
9444     \c_@@_available_keys_str
9445 }
9446 {
9447     The~available~keys~are~(in~alphabetic~order):~
9448     bottomrule,~
9449     code-after,~
9450     code-before,~
9451     detect-duplicates,~
9452     enumitem-keys,~
9453     enumitem-keys-para,~
9454     para,~
9455     label-in-list,~
9456     label-in-tabular~and~
9457     style.
9458 }
9459 \@@_msg_new:nnn { Unknown~key~for~RowStyle }
9460 {
9461     Unknown~key.\\
9462     The~key~'\l_keys_key_str'~is~unknown~for~the~command~
9463     \token_to_str:N \RowStyle. \\
9464     That~key~will~be~ignored. \\
9465     \c_@@_available_keys_str
9466 }
9467 {
9468     The~available~keys~are~(in~alphabetic~order):~
9469     'bold',~
9470     'cell-space-top-limit',~
9471     'cell-space-bottom-limit',~
9472     'cell-space-limits',~
9473     'color',~
9474     'nb-rows'~and~
9475     'rowcolor'.
9476 }
9477 \@@_msg_new:nnn { Unknown~key~for~NiceMatrixOptions }
9478 {
9479     Unknown~key.\\
9480     The~key~'\l_keys_key_str'~is~unknown~for~the~command~
9481     \token_to_str:N \NiceMatrixOptions. \\
9482     That~key~will~be~ignored. \\
9483     \c_@@_available_keys_str
9484 }
9485 {
9486     The~available~keys~are~(in~alphabetic~order):~
9487     allow-duplicate-names,~
9488     caption-above,~
9489     cell-space-bottom-limit,~
9490     cell-space-limits,~
9491     cell-space-top-limit,~
9492     code-for-first-col,~
9493     code-for-first-row,~
9494     code-for-last-col,~

```

```

9495 code-for-last-row,~
9496 corners,~
9497 custom-key,~
9498 create-extra-nodes,~
9499 create-medium-nodes,~
9500 create-large-nodes,~
9501 delimiters~(several~subkeys),~
9502 end-of-row,~
9503 first-col,~
9504 first-row,~
9505 hlines,~
9506 hvlines,~
9507 hvlines-except-borders,~
9508 last-col,~
9509 last-row,~
9510 left-margin,~
9511 light-syntax,~
9512 matrix/columns-type,~
9513 notes~(several~subkeys),~
9514 nullify-dots,~
9515 pgf-node-code,~
9516 renew-dots,~
9517 renew-matrix,~
9518 respect-arraystretch,~
9519 rounded-corners,~
9520 right-margin,~
9521 rules~(with~the~subkeys~'color'~and~'width'),~
9522 small,~
9523 sub-matrix~(several~subkeys),~
9524 vlines,~
9525 xdots~(several~subkeys).
9526 }

```

For ‘{NiceArray}’, the set of keys is the same as for {NiceMatrix} excepted that there is no `l` and `r`.

```

9527 \@@_msg_new:nnn { Unknown~key~for~NiceArray }
9528 {
9529   Unknown~key.\\
9530   The~key~'\l_keys_key_str'~is~unknown~for~the~environment~
9531   \{NiceArray\}. \\
9532   That~key~will~be~ignored. \\
9533   \c_@@_available_keys_str
9534 }
9535 {
9536   The~available~keys~are~(in~alphabetic~order):~
9537   b,~
9538   baseline,~
9539   c,~
9540   cell-space-bottom-limit,~
9541   cell-space-limits,~
9542   cell-space-top-limit,~
9543   code-after,~
9544   code-for-first-col,~
9545   code-for-first-row,~
9546   code-for-last-col,~
9547   code-for-last-row,~
9548   color-inside,~
9549   columns-width,~
9550   corners,~
9551   create-extra-nodes,~
9552   create-medium-nodes,~
9553   create-large-nodes,~
9554   extra-left-margin,~
9555   extra-right-margin,~

```

```

9556     first-col,~
9557     first-row,~
9558     hlines,~
9559     hvlines,~
9560     hvlines-except-borders,~
9561     last-col,~
9562     last-row,~
9563     left-margin,~
9564     light-syntax,~
9565     name,~
9566     nullify-dots,~
9567     pgf-node-code,~
9568     renew-dots,~
9569     respect-arraystretch,~
9570     right-margin,~
9571     rounded-corners,~
9572     rules~(with~the~subkeys~'color'~and~'width'),~
9573     small,~
9574     t,~
9575     vlines,~
9576     xdots/color,~
9577     xdots/shorten-start,~
9578     xdots/shorten-end,~
9579     xdots/shorten-and~
9580     xdots/line-style.
9581 }

```

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`).

```

9582 \@@_msg_new:nnn { Unknown-key-for-NiceMatrix }
9583 {
9584   Unknown~key.\\
9585   The~key~'\l_keys_key_str'~is~unknown~for~the~
9586   \@@_full_name_env:. \\
9587   That~key~will~be~ignored. \\
9588   \c_@@_available_keys_str
9589 }
9590 {
9591   The~available~keys~are~(in~alphabetic~order):~
9592   b,~
9593   baseline,~
9594   c,~
9595   cell-space-bottom-limit,~
9596   cell-space-limits,~
9597   cell-space-top-limit,~
9598   code-after,~
9599   code-for-first-col,~
9600   code-for-first-row,~
9601   code-for-last-col,~
9602   code-for-last-row,~
9603   color-inside,~
9604   columns-type,~
9605   columns-width,~
9606   corners,~
9607   create-extra-nodes,~
9608   create-medium-nodes,~
9609   create-large-nodes,~
9610   extra-left-margin,~
9611   extra-right-margin,~
9612   first-col,~
9613   first-row,~
9614   hlines,~
9615   hvlines,~
9616   hvlines-except-borders,~

```

```

9617 l,~
9618 last-col,~
9619 last-row,~
9620 left-margin,~
9621 light-syntax,~
9622 name,~
9623 nullify-dots,~
9624 pgf-node-code,~
9625 r,~
9626 renew-dots,~
9627 respect-arraystretch,~
9628 right-margin,~
9629 rounded-corners,~
9630 rules~(with~the~subkeys~'color'~and~'width'),~
9631 small,~
9632 t,~
9633 vlines,~
9634 xdots/color,~
9635 xdots/shorten-start,~
9636 xdots/shorten-end,~
9637 xdots/shorten~and~
9638 xdots/line-style.
9639 }
9640 \@@_msg_new:nnn { Unknown~key~for~NiceTabular }
9641 {
9642   Unknown~key.\\
9643   The~key~'\l_keys_key_str'~is~unknown~for~the~environment~
9644   \{NiceTabular\}. \\
9645   That~key~will~be~ignored. \\
9646   \c_@@_available_keys_str
9647 }
9648 {
9649   The~available~keys~are~(in~alphabetic~order):~
9650   b,~
9651   baseline,~
9652   c,~
9653   caption,~
9654   cell-space-bottom-limit,~
9655   cell-space-limits,~
9656   cell-space-top-limit,~
9657   code-after,~
9658   code-for-first-col,~
9659   code-for-first-row,~
9660   code-for-last-col,~
9661   code-for-last-row,~
9662   color-inside,~
9663   columns-width,~
9664   corners,~
9665   custom-line,~
9666   create-extra-nodes,~
9667   create-medium-nodes,~
9668   create-large-nodes,~
9669   extra-left-margin,~
9670   extra-right-margin,~
9671   first-col,~
9672   first-row,~
9673   hlines,~
9674   hvlines,~
9675   hvlines-except-borders,~
9676   label,~
9677   last-col,~
9678   last-row,~
9679   left-margin,~

```

```

9680 light-syntax,~
9681 name,~
9682 notes~(several~subkeys),~
9683 nullify-dots,~
9684 pgf-node-code,~
9685 renew-dots,~
9686 respect-arraystretch,~
9687 right-margin,~
9688 rounded-corners,~
9689 rules~(with~the~subkeys~'color'~and~'width'),~
9690 short-caption,~
9691 t,~
9692 tabularnote,~
9693 vlines,~
9694 xdots/color,~
9695 xdots/shorten-start,~
9696 xdots/shorten-end,~
9697 xdots/shorten-and~
9698 xdots/line-style.
9699 }

9700 \@@_msg_new:nnn { Duplicate-name }
9701 {
9702   Duplicate-name.\
9703   The~name~'\l_keys_value_tl'~is~already~used~and~you~shouldn't~use~
9704   the~same~environment~name~twice.~You~can~go~on,~but,~
9705   maybe,~you~will~have~incorrect~results~especially~
9706   if~you~use~'columns-width=auto'.~If~you~don't~want~to~see~this~
9707   message~again,~use~the~key~'allow-duplicate-names'~in~
9708   '\token_to_str:N \NiceMatrixOptions'.\
9709   \bool_if:NF \g_@@_messages_for_Overleaf_bool
9710   { For~a~list~of~the~names~already~used,~type~H~<return>. }
9711 }
9712 {
9713   The~names~already~defined~in~this~document~are:~
9714   \seq_use:Nnnn \g_@@_names_seq { ~and~ } { ,~ } { ~and~ }.
9715 }

9716 \@@_msg_new:nn { Option-auto-for-columns-width }
9717 {
9718   Erroneous-use.\
9719   You~can't~give~the~value~'auto'~to~the~key~'columns-width'~here.~
9720   That~key~will~be~ignored.
9721 }

```

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	96
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	177
29	Automatic arrays	177
30	The redefinition of the command <code>\dotfill</code>	179

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>	183
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