

The code of the package `nicematrix`*

F. Pantigny
fpantigny@wanadoo.fr

September 28, 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 6.24 of `nicematrix`, at the date of 2023/09/28.

```

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

```

```

284 \cs_new_protected:Npn \@@_rescan_for_spanish:N #1
285 {
286     \tl_set_rescan:Nno
287     #1
288     {
289         \char_set_catcode_other:N >

```



```

290     \char_set_catcode_other:N <
291   }
292   #1
293 }

```

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.

```

294 \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).

```

295 \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.

```

296 \NewExpandableDocumentCommand \NiceMatrixLastEnv { }
297   { \int_use:N \g_@@_env_int }

```

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

```

298 \cs_new_protected:Npn \@@_qpoint:n #1
299   { \pgfpointanchor { \@@_env: - #1 } { center } }

```

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

```

300 \bool_new:N \l_@@_tabular_bool

```

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

```

301 \bool_new:N \g_@@_delims_bool
302 \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.

```

303 \bool_new:N \l_@@_preamble_bool
304 \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.

```

305 \bool_new:N \l_@@_NiceMatrix_without_vlines_bool

```

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

```

306 \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.

```
307 \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).

```
308 \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.).

```
309 \dim_new:N \l_@@_col_width_dim
```

```
310 \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.

```
311 \int_new:N \g_@@_row_total_int
```

```
312 \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).

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

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

```
314 \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.

```
315 \str_new:N \l_@@_hpos_cell_str
```

```
316 \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`.

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

Idem for the mono-row blocks.

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

```
319 \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`.

```
320 \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`).

```
321 \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`.

```
322 \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.

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

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

```
324 \bool_new:N \l_@@_notes_detect_duplicates_bool
325 \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.

```
326 \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).

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

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

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

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

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

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

```
330 \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.

```
331 \bool_new:N \l_@@_X_column_bool
332 \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 }`).

```
333 \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.

```
334 \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.

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

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

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

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

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

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

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

For `\multicolumn`.

```
340 \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`.

```
341 \tl_new:N \l_@@_columns_type_tl
342 \tl_set:Nn \l_@@_columns_type_tl { c }
```

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

```
343 \tl_new:N \l_@@_xdots_down_tl
344 \tl_new:N \l_@@_xdots_up_tl
345 \tl_new:N \l_@@_xdots_middle_tl
```

```
346 \cs_new_protected:Npn \@@_test_if_math_mode:
347 {
348   \if_mode_math: \else:
349     \@@_fatal:n { Outside~math~mode }
350   \fi:
351 }
```

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

```
352 \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.

```
353 \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”.

```
354 \colorlet { nicematrix-last-col } { . }
355 \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`).

```
356 \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*.

```
357 \tl_new:N \g_@@_com_or_env_str
358 \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).

```
359 \cs_new:Npn \@@_full_name_env:
360 {
361   \str_if_eq:VnTF \g_@@_com_or_env_str { command }
362     { command \space \c_backslash_str \g_@@_name_env_str }
363     { environment \space \{ \g_@@_name_env_str \} }
364 }
```

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

```
365 \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.

```
366 \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.

```
367
```

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

```
368 \tl_new:N \g_@@_pre_code_before_tl
```

```
369 \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.

```
370 \tl_new:N \g_@@_pre_code_after_tl
```

```
371 \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.

```
372 \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.

```
373 \int_new:N \l_@@_old_iRow_int
```

```
374 \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).

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

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

```
376 \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.

```
377 \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.

```
378 \bool_new:N \l_@@_X_columns_aux_bool
```

```
379 \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.

```
380 \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).

```
381 \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).

```
382 \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`).

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

```
384 \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).

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

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

```
386 \dim_new:N \l_@@_x_initial_dim
```

```
387 \dim_new:N \l_@@_y_initial_dim
```

```
388 \dim_new:N \l_@@_x_final_dim
```

```
389 \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.

```
390 \dim_zero_new:N \l_@@_tmpc_dim
```

```
391 \dim_zero_new:N \l_@@_tmpd_dim
```

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

```
392 \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”.

```
393 \dim_new:N \g_@@_width_last_col_dim
```

```
394 \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.

```
395 \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!).

```
396 \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: $\{imin\}\{jmin\}\{imax\}\{jmax\}\{name\}$.

```
397 \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.

```
398 \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.

```
399 \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`).

```
400 \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.

```
401 \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).

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

```
403 \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`).

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

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

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

```
407 \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.

```
408 \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).

```
409 \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`.

```
410 \tl_new:N \l_@@_fill_tl
411 \tl_new:N \l_@@_opacity_tl
412 \tl_new:N \l_@@_draw_tl
413 \seq_new:N \l_@@_tikz_seq
414 \clist_new:N \l_@@_borders_clist
415 \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.

```
416 \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`.

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

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

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

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

```
419 \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`).

```
420 \str_new:N \l_@@_hpos_block_str
421 \str_set:Nn \l_@@_hpos_block_str { c }
422 \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`.

```
423 \str_new:N \l_@@_vpos_of_block_str
424 \str_set:Nn \l_@@_vpos_of_block_str { c }
```

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

```
425 \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).

```
426 \bool_new:N \l_@@_vlines_block_bool
427 \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.

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

429 \dim_new:N \l_@@_submatrix_extra_height_dim
430 \dim_new:N \l_@@_submatrix_left_xshift_dim
431 \dim_new:N \l_@@_submatrix_right_xshift_dim
432 \clist_new:N \l_@@_hlines_clist
433 \clist_new:N \l_@@_vlines_clist
434 \clist_new:N \l_@@_submatrix_hlines_clist
435 \clist_new:N \l_@@_submatrix_vlines_clist
```


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

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

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

```
437 \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`).

```
438 \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.

```
439 \int_new:N \l_@@_first_row_int
440 \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.

```
441 \int_new:N \l_@@_first_col_int
442 \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).

```
443 \int_new:N \l_@@_last_row_int
444 \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

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

Idem for `\l_@@_last_col_without_value_bool`

```
446 \bool_new:N \l_@@_last_col_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.

- **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.

```
447 \int_new:N \l_@@_last_col_int
448 \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.

```
449 \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`).

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

Some utilities

```
451 \cs_set_protected:Npn \@@_cut_on_hyphen:w #1-#2\q_stop
452 {
453   \tl_set:Nn \l_tmpa_tl { #1 }
454   \tl_set:Nn \l_tmpb_tl { #2 }
455 }
```

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.

```
456 \cs_new_protected:Npn \@@_expand_clist:N #1
457 {
458   \clist_if_in:NnF #1 { all }
459   {
460     \clist_clear:N \l_tmpa_clist
461     \clist_map_inline:Nn #1
462     {
463       \tl_if_in:nnTF { ##1 } { - }
464       { \@@_cut_on_hyphen:w ##1 \q_stop }
465       {
466         \tl_set:Nn \l_tmpa_tl { ##1 }
467         \tl_set:Nn \l_tmpb_tl { ##1 }
468       }
469       \int_step_inline:nnn { \l_tmpa_tl } { \l_tmpb_tl }
470       { \clist_put_right:Nn \l_tmpa_clist { ####1 } }
471     }
472     \tl_set_eq:NN #1 \l_tmpa_clist
473   }
474 }
```

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.

```

475 \hook_gput_code:nmn { begindocument } { . }
476 {
477   \dim_const:Nn \c_@@_shift_Ldots_last_row_dim { 0.5 em }
478   \dim_const:Nn \c_@@_shift_exterior_Vdots_dim { 0.6 em }
479   \dim_const:Nn \c_@@_innersep_middle_dim { 0.17 em }
480 }

```

6 The command `\tabularnote`

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

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

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

```

481 \newcounter { tabularnote }

```

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

```

482 \seq_new:N \g_@@_notes_seq
483 \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.

```

484 \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.

```

485 \seq_new:N \l_@@_notes_labels_seq
486 \newcounter{nicematrix_draft}
487 \cs_new_protected:Npn \@@_notes_format:n #1
488 {
489   \setcounter { nicematrix_draft } { #1 }
490   \@@_notes_style:n { nicematrix_draft }
491 }

```

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

```

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

```

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

```

493 \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`.

```

494 \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`.

```

495 \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`).

```

496 \hook_gput_code:nnn { begindocument } { . }
497 {
498   \IfPackageLoadedTF { enumitem }
499   {

```

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.

```

500   \newlist { tabularnotes } { enumerate } { 1 }
501   \setlist [ tabularnotes ]
502   {
503     topsep = 0pt ,
504     noitemsep ,
505     leftmargin = * ,
506     align = left ,
507     labelsep = 0pt ,
508     label =
509       \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotesi } } ,
510   }
511   \newlist { tabularnotes* } { enumerate* } { 1 }
512   \setlist [ tabularnotes* ]
513   {
514     afterlabel = \nobreak ,

```

```

515         itemjoin = \quad ,
516         label =
517         \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotes*i } }
518     }

```

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`.

```

519     \NewDocumentCommand \tabularnote { o m }
520     {
521         \bool_if:nT { \cs_if_exist_p:N \@captive || \l_@@_in_env_bool }
522         {
523             \bool_if:nTF { ! \l_@@_tabular_bool && \l_@@_in_env_bool }
524             { \@@_error:n { tabularnote~forbidden } }
525             {
526                 \bool_if:nTF \l_@@_in_caption_bool
527                 { \@@_tabularnote_caption:nn { #1 } { #2 } }
528                 { \@@_tabularnote:nn { #1 } { #2 } }
529             }
530         }
531     }
532 }
533 {
534     \NewDocumentCommand \tabularnote { o m }
535     {
536         \@@_error_or_warning:n { enumitem~not~loaded }
537         \@@_gredirect_none:n { enumitem~not~loaded }
538     }
539 }
540 }

```

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`.

```

541 \cs_new_protected:Npn \@@_tabularnote:nn #1 #2
542 {

```

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).

```

543     \int_zero:N \l_tmpa_int
544     \bool_if:NT \l_@@_notes_detect_duplicates_bool
545     {

```

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`.

```

546     \seq_map_indexed_inline:Nn \g_@@_notes_seq
547     {
548         \tl_if_eq:nnT { { #1 } { #2 } } { ##2 }
549         {
550             \int_set:Nn \l_tmpa_int { ##1 }
551             \seq_map_break:
552         }
553     }
554     \int_compare:nNf \l_tmpa_int = \c_zero_int
555     { \int_add:Nn \l_tmpa_int \g_@@_notes_caption_int }

```

```

556     }
557     \int_compare:nNnT \l_tmpa_int = \c_zero_int
558     {
559         \seq_gput_right:Nn \g_@@_notes_seq { { #1 } { #2 } }
560         \tl_if_novalue:nT { #1 } { \int_gincr:N \c@tabularnote }
561     }
562     \seq_put_right:Nx \l_@@_notes_labels_seq
563     {
564         \tl_if_novalue:nTF { #1 }
565         {
566             \@@_notes_format:n
567             {
568                 \int_eval:n
569                 {
570                     \int_compare:nNnTF \l_tmpa_int = \c_zero_int
571                     \c@tabularnote
572                     \l_tmpa_int
573                 }
574             }
575         }
576         { #1 }
577     }
578     \peek_meaning:NF \tabularnote
579     {

```

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`.

```

580         \hbox_set:Nn \l_tmpa_box
581         {

```

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

```

582             \@@_notes_label_in_tabular:n
583             {
584                 \seq_use:Nnnn
585                 \l_@@_notes_labels_seq { , } { , } { , }
586             }
587         }

```

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

```

588         \int_gsub:Nn \c@tabularnote { 1 }
589         \int_set_eq:NN \l_tmpa_int \c@tabularnote
590         \refstepcounter { tabularnote }
591         \int_compare:nNnT \l_tmpa_int = \c@tabularnote
592         { \int_gincr:N \c@tabularnote }
593         \seq_clear:N \l_@@_notes_labels_seq
594         \bool_lazy_or:nnTF
595         { \str_if_eq_p:Vn \l_@@_hpos_cell_str { c } }
596         { \str_if_eq_p:Vn \l_@@_hpos_cell_str { r } }
597         {
598             \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).

```

599             \skip_horizontal:n { \box_wd:N \l_tmpa_box }
600         }
601         { \box_use:N \l_tmpa_box }
602     }
603 }

```

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`.

```

604 \cs_new_protected:Npn \@@_tabularnote_caption:nn #1 #2
605 {
606   \bool_if:NTF \g_@@_caption_finished_bool
607   {
608     \int_compare:nNnT
609     \c@tabularnote = \g_@@_notes_caption_int
610     { \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`!

```

611   \seq_if_in:NnF \g_@@_notes_in_caption_seq { { #1 } { #2 } }
612   { \@@_error:n { Identical~notes~in~caption } }
613 }
614 {

```

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

```

615   \seq_if_in:NnTF \g_@@_notes_in_caption_seq { { #1 } { #2 } }
616   {

```

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.

```

617     \bool_gset_true:N \g_@@_caption_finished_bool
618     \int_gset_eq:NN \g_@@_notes_caption_int \c@tabularnote
619     \int_gzero:N \c@tabularnote
620   }
621   { \seq_gput_right:Nn \g_@@_notes_in_caption_seq { { #1 } { #2 } } }
622 }

```

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!

```

623   \tl_if_novalue:nT { #1 } { \int_gincr:N \c@tabularnote }
624   \seq_put_right:Nx \l_@@_notes_labels_seq
625   {
626     \tl_if_novalue:nTF { #1 }
627     { \@@_notes_format:n { \int_use:N \c@tabularnote } }
628     { #1 }
629   }
630   \peek_meaning:NF \tabularnote
631   {
632     \@@_notes_label_in_tabular:n
633     { \seq_use:Nnnn \l_@@_notes_labels_seq { , } { , } { , } }
634     \seq_clear:N \l_@@_notes_labels_seq
635   }
636 }

637 \cs_new_protected:Npn \@@_count_novalue_first:nn #1 #2
638 { \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.

```

639 \cs_new_protected:Npn \@@_pgf_rect_node:nnnnn #1 #2 #3 #4 #5
640 {
641   \begin { pgfscope }
642   \pgfset
643   {
644     inner~sep = \c_zero_dim ,
645     minimum~size = \c_zero_dim
646   }
647   \pgftransformshift { \pgfpoint { 0.5 * ( #2 + #4 ) } { 0.5 * ( #3 + #5 ) } }
648   \pgfnode
649   { rectangle }
650   { center }
651   {
652     \vbox_to_ht:nn
653     { \dim_abs:n { #5 - #3 } }
654     {
655       \vfill
656       \hbox_to_wd:nn { \dim_abs:n { #4 - #2 } } { }
657     }
658   }
659   { #1 }
660   { }
661   \end { pgfscope }
662 }

```

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.

```

663 \cs_new_protected:Npn \@@_pgf_rect_node:nnn #1 #2 #3
664 {
665   \begin { pgfscope }
666   \pgfset
667   {
668     inner~sep = \c_zero_dim ,
669     minimum~size = \c_zero_dim
670   }
671   \pgftransformshift { \pgfpoint scale { 0.5 } { \pgfpointadd { #2 } { #3 } } }
672   \pgfpointdiff { #3 } { #2 }
673   \pgfgetlastxy \l_tmpa_dim \l_tmpb_dim
674   \pgfnode
675   { rectangle }
676   { center }
677   {
678     \vbox_to_ht:nn
679     { \dim_abs:n \l_tmpb_dim }
680     { \vfill \hbox_to_wd:nn { \dim_abs:n \l_tmpa_dim } { } }
681   }
682   { #1 }
683   { }
684   \end { pgfscope }
685 }

```

8 The options

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

```

686 \tl_new:N \l_@@_caption_tl

```



```

687 \tl_new:N \l_@@_short_caption_tl
688 \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).

```

689 \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.

```

690 \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`.

```

691 \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`).

```

692 \dim_new:N \l_@@_cell_space_top_limit_dim
693 \dim_new:N \l_@@_cell_space_bottom_limit_dim

```

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

```

694 \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.

```

695 \dim_new:N \l_@@_xdots_inter_dim
696 \hook_gput_code:nnn { begindocument } { . }
697 { \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).

```

698 \dim_new:N \l_@@_xdots_shorten_start_dim
699 \dim_new:N \l_@@_xdots_shorten_end_dim
700 \hook_gput_code:nnn { begindocument } { . }
701 {
702   \dim_set:Nn \l_@@_xdots_shorten_start_dim { 0.3 em }
703   \dim_set:Nn \l_@@_xdots_shorten_end_dim { 0.3 em }
704 }

```

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.

```

705 \dim_new:N \l_@@_xdots_radius_dim
706 \hook_gput_code:nnn { begindocument } { . }
707 { \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.

```
708 \tl_new:N \l_@@_xdots_line_style_tl
709 \tl_const:Nn \c_@@_standard_tl { standard }
710 \tl_set_eq:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl
```

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

```
711 \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).

```
712 \tl_new:N \l_@@_baseline_tl
713 \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`).

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

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

```
715 \bool_new:N \l_@@_parallelize_diags_bool
716 \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.

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

```
718 \dim_new:N \l_@@_notes_above_space_dim
719 \hook_gput_code:nnn { begindocument } { . }
720 { \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.).

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

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

```
722 \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).

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

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

```
724 \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.

```
725 \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”.

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

```
727 \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).

```
728 \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`).

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

```
730 \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`.

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

```
732 \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.

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

```
734 \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 “:”.

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

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

```
736 \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.

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

```
738 \keys_define:nn { NiceMatrix / xdots }
739 {
740   shorten-start .code:n =
741     \hook_gput_code:nnn { begindocument } { . }
742     { \dim_set:Nn \l_@@_xdots_shorten_start_dim { #1 } } ,
743   shorten-end .code:n =
744     \hook_gput_code:nnn { begindocument } { . }
745     { \dim_set:Nn \l_@@_xdots_shorten_end_dim { #1 } } ,
746   shorten-start .value_required:n = true ,
747   shorten-end .value_required:n = true ,
748   shorten .code:n =
749     \hook_gput_code:nnn { begindocument } { . }
```

```

750     {
751         \dim_set:Nn \l_@@_xdots_shorten_start_dim { #1 }
752         \dim_set:Nn \l_@@_xdots_shorten_end_dim { #1 }
753     } ,
754     shorten .value_required:n = true ,
755     horizontal-labels .bool_set:N = \l_@@_xdots_h_labels_bool ,
756     horizontal-labels .default:n = true ,
757     line-style .code:n =
758     {
759         \bool_lazy_or:nnTF
760         { \cs_if_exist_p:N \tikzpicture }
761         { \str_if_eq_p:nn { #1 } { standard } }
762         { \tl_set:Nn \l_@@_xdots_line_style_tl { #1 } }
763         { \@@_error:n { bad-option-for-line-style } }
764     } ,
765     line-style .value_required:n = true ,
766     color .tl_set:N = \l_@@_xdots_color_tl ,
767     color .value_required:n = true ,
768     radius .code:n =
769     { \hook_gput_code:nnn { begindocument } { . }
770       { \dim_set:Nn \l_@@_xdots_radius_dim { #1 } } ,
771     radius .value_required:n = true ,
772     inter .code:n =
773     { \hook_gput_code:nnn { begindocument } { . }
774       { \dim_set:Nn \l_@@_xdots_inter_dim { #1 } } } ,
775     radius .value_required:n = true ,

```

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

```

776     down .code:n = \tl_put_right:Nn \l_@@_xdots_down_tl { #1 } , % modified 2023-08-09
777     up .code:n = \tl_put_right:Nn \l_@@_xdots_up_tl { #1 } ,
778     middle .code:n = \tl_put_right:Nn \l_@@_xdots_middle_tl { #1 } ,

```

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

```

779     draw-first .code:n = \prg_do_nothing: ,
780     unknown .code:n = \@@_error:n { Unknown-key-for-xdots }
781 }

```

```

782 \keys_define:nn { NiceMatrix / rules }
783 {
784     color .tl_set:N = \l_@@_rules_color_tl ,
785     color .value_required:n = true ,
786     width .dim_set:N = \arrayrulewidth ,
787     width .value_required:n = true ,
788     unknown .code:n = \@@_error:n { Unknown-key-for-rules }
789 }

```

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

```

790 \keys_define:nn { NiceMatrix / Global }
791 {
792     rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
793     rounded-corners .default:n = 4 pt ,
794     custom-line .code:n = \@@_custom_line:n { #1 } ,
795     rules .code:n = \keys_set:nn { NiceMatrix / rules } { #1 } ,
796     rules .value_required:n = true ,
797     standard-cline .bool_set:N = \l_@@_standard_cline_bool ,
798     standard-cline .default:n = true ,
799     cell-space-top-limit .dim_set:N = \l_@@_cell_space_top_limit_dim ,

```

```

800 cell-space-top-limit .value_required:n = true ,
801 cell-space-bottom-limit .dim_set:N = \l_@@_cell_space_bottom_limit_dim ,
802 cell-space-bottom-limit .value_required:n = true ,
803 cell-space-limits .meta:n =
804 {
805     cell-space-top-limit = #1 ,
806     cell-space-bottom-limit = #1 ,
807 } ,
808 cell-space-limits .value_required:n = true ,
809 xdots .code:n = \keys_set:nn { NiceMatrix / xdots } { #1 } ,
810 light-syntax .bool_set:N = \l_@@_light_syntax_bool ,
811 light-syntax .default:n = true ,
812 end-of-row .tl_set:N = \l_@@_end_of_row_tl ,
813 end-of-row .value_required:n = true ,
814 first-col .code:n = \int_zero:N \l_@@_first_col_int ,
815 first-row .code:n = \int_zero:N \l_@@_first_row_int ,
816 last-row .int_set:N = \l_@@_last_row_int ,
817 last-row .default:n = -1 ,
818 code-for-first-col .tl_set:N = \l_@@_code_for_first_col_tl ,
819 code-for-first-col .value_required:n = true ,
820 code-for-last-col .tl_set:N = \l_@@_code_for_last_col_tl ,
821 code-for-last-col .value_required:n = true ,
822 code-for-first-row .tl_set:N = \l_@@_code_for_first_row_tl ,
823 code-for-first-row .value_required:n = true ,
824 code-for-last-row .tl_set:N = \l_@@_code_for_last_row_tl ,
825 code-for-last-row .value_required:n = true ,
826 hlines .clist_set:N = \l_@@_hlines_clist ,
827 vlines .clist_set:N = \l_@@_vlines_clist ,
828 hlines .default:n = all ,
829 vlines .default:n = all ,
830 vlines-in-sub-matrix .code:n =
831 {
832     \tl_if_single_token:nTF { #1 }
833     { \tl_set:Nn \l_@@_letter_vlism_tl { #1 } }
834     { \@@_error:n { One-letter-allowed } }
835 } ,
836 vlines-in-sub-matrix .value_required:n = true ,
837 hvlines .code:n =
838 {
839     \bool_set_true:N \l_@@_hvlines_bool
840     \clist_set:Nn \l_@@_vlines_clist { all }
841     \clist_set:Nn \l_@@_hlines_clist { all }
842 } ,
843 hvlines-except-borders .code:n =
844 {
845     \clist_set:Nn \l_@@_vlines_clist { all }
846     \clist_set:Nn \l_@@_hlines_clist { all }
847     \bool_set_true:N \l_@@_hvlines_bool
848     \bool_set_true:N \l_@@_except_borders_bool
849 } ,
850 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.

```

851 renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
852 renew-dots .value_forbidden:n = true ,
853 nullify-dots .bool_set:N = \l_@@_nullify_dots_bool ,
854 create-medium-nodes .bool_set:N = \l_@@_medium_nodes_bool ,
855 create-large-nodes .bool_set:N = \l_@@_large_nodes_bool ,
856 create-extra-nodes .meta:n =
857 { create-medium-nodes , create-large-nodes } ,
858 left-margin .dim_set:N = \l_@@_left_margin_dim ,
859 left-margin .default:n = \arraycolsep ,

```

```

860 right-margin .dim_set:N = \l_@@_right_margin_dim ,
861 right-margin .default:n = \arraycolsep ,
862 margin .meta:n = { left-margin = #1 , right-margin = #1 } ,
863 margin .default:n = \arraycolsep ,
864 extra-left-margin .dim_set:N = \l_@@_extra_left_margin_dim ,
865 extra-right-margin .dim_set:N = \l_@@_extra_right_margin_dim ,
866 extra-margin .meta:n =
867   { extra-left-margin = #1 , extra-right-margin = #1 } ,
868 extra-margin .value_required:n = true ,
869 respect-arraystretch .bool_set:N = \l_@@_respect_arraystretch_bool ,
870 respect-arraystretch .default:n = true ,
871 pgf-node-code .tl_set:N = \l_@@_pgf_node_code_tl ,
872 pgf-node-code .value_required:n = true
873 }

```

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

```

874 \keys_define:nn { NiceMatrix / Env }
875 {
876   corners .clist_set:N = \l_@@_corners_clist ,
877   corners .default:n = { NW , SW , NE , SE } ,
878   code-before .code:n =
879     {
880       \tl_if_empty:nF { #1 }
881       {
882         \tl_gput_left:Nn \g_@@_pre_code_before_tl { #1 }
883         \bool_set_true:N \l_@@_code_before_bool
884       }
885     } ,
886   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}`.

```

887   c .code:n = \tl_set:Nn \l_@@_baseline_tl c ,
888   t .code:n = \tl_set:Nn \l_@@_baseline_tl t ,
889   b .code:n = \tl_set:Nn \l_@@_baseline_tl b ,
890   baseline .tl_set:N = \l_@@_baseline_tl ,
891   baseline .value_required:n = true ,
892   columns-width .code:n =
893     \tl_if_eq:nnTF { #1 } { auto }
894     { \bool_set_true:N \l_@@_auto_columns_width_bool }
895     { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,
896   columns-width .value_required:n = true ,
897   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.

```

898   \legacy_if:nF { measuring@ }
899   {
900     \str_set:Nx \l_tmpa_str { #1 }
901     \seq_if_in:NVTF \g_@@_names_seq \l_tmpa_str
902     { \@@_error:nn { Duplicate~name } { #1 } }
903     { \seq_gput_left:NV \g_@@_names_seq \l_tmpa_str }
904     \str_set_eq:NN \l_@@_name_str \l_tmpa_str
905   } ,
906   name .value_required:n = true ,
907   code-after .tl_gset:N = \g_nicematrix_code_after_tl ,
908   code-after .value_required:n = true ,
909   color-inside .code:n =
910     \bool_set_true:N \l_@@_color_inside_bool
911     \bool_set_true:N \l_@@_code_before_bool ,
912   color-inside .value_forbidden:n = true ,

```

```

913     colortbl-like .meta:n = color-inside
914 }
915 \keys_define:nn { NiceMatrix / notes }
916 {
917     para .bool_set:N = \l_@@_notes_para_bool ,
918     para .default:n = true ,
919     code-before .tl_set:N = \l_@@_notes_code_before_tl ,
920     code-before .value_required:n = true ,
921     code-after .tl_set:N = \l_@@_notes_code_after_tl ,
922     code-after .value_required:n = true ,
923     bottomrule .bool_set:N = \l_@@_notes_bottomrule_bool ,
924     bottomrule .default:n = true ,
925     style .code:n = \cs_set:Nn \@@_notes_style:n { #1 } ,
926     style .value_required:n = true ,
927     label-in-tabular .code:n =
928         \cs_set:Nn \@@_notes_label_in_tabular:n { #1 } ,
929     label-in-tabular .value_required:n = true ,
930     label-in-list .code:n =
931         \cs_set:Nn \@@_notes_label_in_list:n { #1 } ,
932     label-in-list .value_required:n = true ,
933     enumitem-keys .code:n =
934         {
935             \hook_gput_code:nnn { begindocument } { . }
936             {
937                 \IfPackageLoadedTF { enumitem }
938                 { \setlist* [ tabularnotes ] { #1 } }
939                 { }
940             }
941         } ,
942     enumitem-keys .value_required:n = true ,
943     enumitem-keys-para .code:n =
944         {
945             \hook_gput_code:nnn { begindocument } { . }
946             {
947                 \IfPackageLoadedTF { enumitem }
948                 { \setlist* [ tabularnotes* ] { #1 } }
949                 { }
950             }
951         } ,
952     enumitem-keys-para .value_required:n = true ,
953     detect-duplicates .bool_set:N = \l_@@_notes_detect_duplicates_bool ,
954     detect-duplicates .default:n = true ,
955     unknown .code:n = \@@_error:n { Unknown~key~for~notes }
956 }
957 \keys_define:nn { NiceMatrix / delimiters }
958 {
959     max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
960     max-width .default:n = true ,
961     color .tl_set:N = \l_@@_delimiters_color_tl ,
962     color .value_required:n = true ,
963 }

```

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

```

964 \keys_define:nn { NiceMatrix }
965 {
966     NiceMatrixOptions .inherit:n =
967         { NiceMatrix / Global } ,
968     NiceMatrixOptions / xdots .inherit:n = NiceMatrix / xdots ,
969     NiceMatrixOptions / rules .inherit:n = NiceMatrix / rules ,
970     NiceMatrixOptions / notes .inherit:n = NiceMatrix / notes ,
971     NiceMatrixOptions / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,

```

```

972 SubMatrix / rules .inherit:n = NiceMatrix / rules ,
973 CodeAfter / xdots .inherit:n = NiceMatrix / xdots ,
974 CodeBefore / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
975 CodeAfter / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
976 NiceMatrix .inherit:n =
977 {
978     NiceMatrix / Global ,
979     NiceMatrix / Env ,
980 } ,
981 NiceMatrix / xdots .inherit:n = NiceMatrix / xdots ,
982 NiceMatrix / rules .inherit:n = NiceMatrix / rules ,
983 NiceTabular .inherit:n =
984 {
985     NiceMatrix / Global ,
986     NiceMatrix / Env
987 } ,
988 NiceTabular / xdots .inherit:n = NiceMatrix / xdots ,
989 NiceTabular / rules .inherit:n = NiceMatrix / rules ,
990 NiceTabular / notes .inherit:n = NiceMatrix / notes ,
991 NiceArray .inherit:n =
992 {
993     NiceMatrix / Global ,
994     NiceMatrix / Env ,
995 } ,
996 NiceArray / xdots .inherit:n = NiceMatrix / xdots ,
997 NiceArray / rules .inherit:n = NiceMatrix / rules ,
998 pNiceArray .inherit:n =
999 {
1000     NiceMatrix / Global ,
1001     NiceMatrix / Env ,
1002 } ,
1003 pNiceArray / xdots .inherit:n = NiceMatrix / xdots ,
1004 pNiceArray / rules .inherit:n = NiceMatrix / rules ,
1005 }

```

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

```

1006 \keys_define:nn { NiceMatrix / NiceMatrixOptions }
1007 {
1008     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1009     delimiters / color .value_required:n = true ,
1010     delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1011     delimiters / max-width .default:n = true ,
1012     delimiters .code:n = \keys_set:nn { NiceMatrix / delimiters } { #1 } ,
1013     delimiters .value_required:n = true ,
1014     width .code:n = \dim_set:Nn \l_@@_width_dim { #1 } ,
1015     width .value_required:n = true ,
1016     last-col .code:n =
1017         \tl_if_empty:nF { #1 }
1018         { \@@_error:n { last-col-non-empty-for-NiceMatrixOptions } }
1019         \int_zero:N \l_@@_last_col_int ,
1020     small .bool_set:N = \l_@@_small_bool ,
1021     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.

```

1022     renew-matrix .code:n = \@@_renew_matrix: ,
1023     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}`.

```

1024     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.

```

1025   columns-width .code:n =
1026     \tl_if_eq:nnTF { #1 } { auto }
1027     { \@@_error:n { Option~auto~for~columns~width } }
1028     { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,

```

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

```

1029   allow-duplicate-names .code:n =
1030     \@@_msg_redirect_name:nn { Duplicate-name } { none } ,
1031   allow-duplicate-names .value_forbidden:n = true ,
1032   notes .code:n = \keys_set:nn { NiceMatrix / notes } { #1 } ,
1033   notes .value_required:n = true ,
1034   sub-matrix .code:n = \keys_set:nn { NiceMatrix / sub-matrix } { #1 } ,
1035   sub-matrix .value_required:n = true ,
1036   matrix / columns-type .tl_set:N = \l_@@_columns_type_tl ,
1037   matrix / columns-type .value_required:n = true ,
1038   caption-above .bool_set:N = \l_@@_caption_above_bool ,
1039   caption-above .default:n = true ,
1040   unknown .code:n = \@@_error:n { Unknown~key~for~NiceMatrixOptions }
1041 }

```

`\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.

```

1042 \NewDocumentCommand \NiceMatrixOptions { m }
1043 { \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.

```

1044 \keys_define:nn { NiceMatrix / NiceMatrix }
1045 {
1046   last-col .code:n = \tl_if_empty:nTF {#1}
1047   {
1048     \bool_set_true:N \l_@@_last_col_without_value_bool
1049     \int_set:Nn \l_@@_last_col_int { -1 }
1050   }
1051   { \int_set:Nn \l_@@_last_col_int { #1 } } ,
1052   columns-type .tl_set:N = \l_@@_columns_type_tl ,
1053   columns-type .value_required:n = true ,
1054   l .meta:n = { columns-type = l } ,
1055   r .meta:n = { columns-type = r } ,
1056   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1057   delimiters / color .value_required:n = true ,
1058   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1059   delimiters / max-width .default:n = true ,
1060   delimiters .code:n = \keys_set:nn { NiceMatrix / delimiters } { #1 } ,
1061   delimiters .value_required:n = true ,
1062   small .bool_set:N = \l_@@_small_bool ,
1063   small .value_forbidden:n = true ,
1064   unknown .code:n = \@@_error:n { Unknown~key~for~NiceMatrix }
1065 }

```

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

```

1066 \keys_define:nn { NiceMatrix / NiceArray }
1067 {

```

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.

```

1068   small .bool_set:N = \l_@@_small_bool ,
1069   small .value_forbidden:n = true ,
1070   last-col .code:n = \tl_if_empty:nF { #1 }
1071                     { \@@_error:n { last-col~non~empty~for~NiceArray } }
1072                     \int_zero:N \l_@@_last_col_int ,
1073   r .code:n = \@@_error:n { r~or~l~with~preamble } ,
1074   l .code:n = \@@_error:n { r~or~l~with~preamble } ,
1075   unknown .code:n = \@@_error:n { Unknown~key~for~NiceArray }
1076 }

1077 \keys_define:nn { NiceMatrix / pNiceArray }
1078 {
1079   first-col .code:n = \int_zero:N \l_@@_first_col_int ,
1080   last-col .code:n = \tl_if_empty:nF {#1}
1081                     { \@@_error:n { last-col~non~empty~for~NiceArray } }
1082                     \int_zero:N \l_@@_last_col_int ,
1083   first-row .code:n = \int_zero:N \l_@@_first_row_int ,
1084   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1085   delimiters / color .value_required:n = true ,
1086   delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1087   delimiters / max-width .default:n = true ,
1088   delimiters .code:n = \keys_set:nn { NiceMatrix / delimiters } { #1 } ,
1089   delimiters .value_required:n = true ,
1090   small .bool_set:N = \l_@@_small_bool ,
1091   small .value_forbidden:n = true ,
1092   r .code:n = \@@_error:n { r~or~l~with~preamble } ,
1093   l .code:n = \@@_error:n { r~or~l~with~preamble } ,
1094   unknown .code:n = \@@_error:n { Unknown~key~for~NiceMatrix }
1095 }

```

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

```

1096 \keys_define:nn { NiceMatrix / NiceTabular }
1097 {

```

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.

```

1098   width .code:n = \dim_set:Nn \l_@@_width_dim { #1 }
1099                     \bool_set_true:N \l_@@_width_used_bool ,
1100   width .value_required:n = true ,
1101   notes .code:n = \keys_set:nn { NiceMatrix / notes } { #1 } ,
1102   tabularnote .tl_gset:N = \g_@@_tabularnote_tl ,
1103   tabularnote .value_required:n = true ,
1104   caption .tl_set:N = \l_@@_caption_tl ,
1105   caption .value_required:n = true ,
1106   short-caption .tl_set:N = \l_@@_short_caption_tl ,
1107   short-caption .value_required:n = true ,
1108   label .tl_set:N = \l_@@_label_tl ,
1109   label .value_required:n = true ,
1110   last-col .code:n = \tl_if_empty:nF {#1}
1111                     { \@@_error:n { last-col~non~empty~for~NiceArray } }
1112                     \int_zero:N \l_@@_last_col_int ,
1113   r .code:n = \@@_error:n { r~or~l~with~preamble } ,
1114   l .code:n = \@@_error:n { r~or~l~with~preamble } ,
1115   unknown .code:n = \@@_error:n { Unknown~key~for~NiceTabular }
1116 }

```

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

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

```

1117 \keys_define:nn { NiceMatrix / CodeAfter }
1118 {
1119   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1120   delimiters / color .value_required:n = true ,
1121   rules .code:n = \keys_set:nn { NiceMatrix / rules } { #1 } ,
1122   rules .value_required:n = true ,
1123   xdots .code:n = \keys_set:nn { NiceMatrix / xdots } { #1 } ,
1124   sub-matrix .code:n = \keys_set:nn { NiceMatrix / sub-matrix } { #1 } ,
1125   sub-matrix .value_required:n = true ,
1126   unknown .code:n = \@@_error:n { Unknown~key~for~CodeAfter }
1127 }

```

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}`).

```

1128 \cs_new_protected:Npn \@@_cell_begin:w
1129 {

```

`\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.

```
1130   \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).

```
1131   \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
```

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

```
1132   \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.

```

1133   \int_compare:nNnT \c@jCol = 1
1134   { \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).

```

1135   \hbox_set:Nw \l_@@_cell_box
1136   \bool_if:NF \l_@@_tabular_bool
1137   {
1138     \c_math_toggle_token
1139     \bool_if:NT \l_@@_small_bool \scriptstyle
1140   }
1141   \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.

```

1142   \int_compare:nNnTF \c@iRow = 0
1143   {
1144     \int_compare:nNnT \c@jCol > 0
1145     {
1146       \l_@@_code_for_first_row_tl
1147       \xglobal \colorlet { nicematrix-first-row } { . }
1148     }

```

```

1149     }
1150     {
1151         \int_compare:nNnT \c@iRow = \l_@@_last_row_int
1152         {
1153             \l_@@_code_for_last_row_tl
1154             \xglobal \colorlet { nicematrix-last-row } { . }
1155         }
1156     }
1157 }

```

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.

```

1158 \cs_new_protected:Npn \@@_begin_of_row:
1159 {
1160     \int_gincr:N \c@iRow
1161     \dim_gset_eq:NN \g_@@_dp_ante_last_row_dim \g_@@_dp_last_row_dim
1162     \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
1163     \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1164     \pgfpicture
1165     \pgfrememberpicturepositiononpagetrue
1166     \pgfcoordinate
1167     { \@@_env: - row - \int_use:N \c@iRow - base }
1168     { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
1169     \str_if_empty:NF \l_@@_name_str
1170     {
1171         \pgfnodealias
1172         { \l_@@_name_str - row - \int_use:N \c@iRow - base }
1173         { \@@_env: - row - \int_use:N \c@iRow - base }
1174     }
1175     \endpgfpicture
1176 }

```

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.

```

1177 \cs_new_protected:Npn \@@_update_for_first_and_last_row:
1178 {
1179     \int_compare:nNnTF \c@iRow = 0
1180     {
1181         \dim_gset:Nn \g_@@_dp_row_zero_dim
1182         { \dim_max:nn \g_@@_dp_row_zero_dim { \box_dp:N \l_@@_cell_box } }
1183         \dim_gset:Nn \g_@@_ht_row_zero_dim
1184         { \dim_max:nn \g_@@_ht_row_zero_dim { \box_ht:N \l_@@_cell_box } }
1185     }
1186     {
1187         \int_compare:nNnT \c@iRow = 1
1188         {
1189             \dim_gset:Nn \g_@@_ht_row_one_dim
1190             { \dim_max:nn \g_@@_ht_row_one_dim { \box_ht:N \l_@@_cell_box } }
1191         }
1192     }
1193 }
1194 \cs_new_protected:Npn \@@_rotate_cell_box:
1195 {
1196     \box_rotate:Nn \l_@@_cell_box { 90 }
1197     \bool_if:NTF \g_@@_rotate_c_bool
1198     {
1199         \hbox_set:Nn \l_@@_cell_box

```

```

1200     {
1201         \c_math_toggle_token
1202         \vcenter { \box_use:N \l_@@_cell_box }
1203         \c_math_toggle_token
1204     }
1205 }
1206 {
1207     \int_compare:nNnT \c@iRow = \l_@@_last_row_int
1208     {
1209         \vbox_set_top:Nn \l_@@_cell_box
1210         {
1211             \vbox_to_zero:n { }
1212             \skip_vertical:n { - \box_ht:N \@arstrutbox + 0.8 ex }
1213             \box_use:N \l_@@_cell_box
1214         }
1215     }
1216 }
1217 \bool_gset_false:N \g_@@_rotate_bool
1218 \bool_gset_false:N \g_@@_rotate_c_bool
1219 }
1220 \cs_new_protected:Npn \@@_adjust_size_box:
1221 {
1222     \dim_compare:nNnT \g_@@_blocks_wd_dim > \c_zero_dim
1223     {
1224         \box_set_wd:Nn \l_@@_cell_box
1225         { \dim_max:nn { \box_wd:N \l_@@_cell_box } \g_@@_blocks_wd_dim }
1226         \dim_gzero:N \g_@@_blocks_wd_dim
1227     }
1228     \dim_compare:nNnT \g_@@_blocks_dp_dim > \c_zero_dim
1229     {
1230         \box_set_dp:Nn \l_@@_cell_box
1231         { \dim_max:nn { \box_dp:N \l_@@_cell_box } \g_@@_blocks_dp_dim }
1232         \dim_gzero:N \g_@@_blocks_dp_dim
1233     }
1234     \dim_compare:nNnT \g_@@_blocks_ht_dim > \c_zero_dim
1235     {
1236         \box_set_ht:Nn \l_@@_cell_box
1237         { \dim_max:nn { \box_ht:N \l_@@_cell_box } \g_@@_blocks_ht_dim }
1238         \dim_gzero:N \g_@@_blocks_ht_dim
1239     }
1240 }
1241 \cs_new_protected:Npn \@@_cell_end:
1242 {
1243     \@@_math_toggle_token:
1244     \hbox_set_end:
1245     \@@_cell_end_i:
1246 }
1247 \cs_new_protected:Npn \@@_cell_end_i:
1248 {

```

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.

```

1249     \g_@@_cell_after_hook_tl
1250     \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
1251     \@@_adjust_size_box:
1252     \box_set_ht:Nn \l_@@_cell_box
1253     { \box_ht:N \l_@@_cell_box + \l_@@_cell_space_top_limit_dim }
1254     \box_set_dp:Nn \l_@@_cell_box
1255     { \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”).

```

1256 \dim_gset:Nn \g_@@_max_cell_width_dim
1257 { \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”.

```

1258 \@@_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.

```

1259 \bool_if:NTF \g_@@_empty_cell_bool
1260 { \box_use_drop:N \l_@@_cell_box }
1261 {
1262   \bool_lazy_or:nnTF
1263     \g_@@_not_empty_cell_bool
1264     { \dim_compare_p:nNn { \box_wd:N \l_@@_cell_box } > \c_zero_dim }
1265     \@@_node_for_cell:
1266     { \box_use_drop:N \l_@@_cell_box }
1267 }
1268 \int_gset:Nn \g_@@_col_total_int { \int_max:nn \g_@@_col_total_int \c_jCol }
1269 \bool_gset_false:N \g_@@_empty_cell_bool
1270 \bool_gset_false:N \g_@@_not_empty_cell_bool
1271 }

```

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`).

```

1272 \cs_new_protected:Npn \@@_cell_end_for_w_s:
1273 {
1274   \@@_math_toggle_token:
1275   \hbox_set_end:
1276   \bool_if:NF \g_@@_rotate_bool
1277   {
1278     \hbox_set:Nn \l_@@_cell_box
1279     {
1280       \makebox [ \l_@@_col_width_dim ] [ s ]
1281       { \hbox_unpack_drop:N \l_@@_cell_box }
1282     }
1283   }
1284   \@@_cell_end_i:
1285 }

```

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

```

1286 \pgfset

```

```

1287 {
1288   nicematrix / cell-node /.style =
1289   {
1290     inner~sep = \c_zero_dim ,
1291     minimum~width = \c_zero_dim
1292   }
1293 }
1294 \cs_new_protected:Npn \@@_node_for_cell:
1295 {
1296   \pgfpicture
1297   \pgfsetbaseline \c_zero_dim
1298   \pgfrememberpicturepositiononpagetrue
1299   \pgfset { nicematrix / cell-node }
1300   \pgfnode
1301     { rectangle }
1302     { base }
1303     {

```

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).

```

1304   \set@color
1305   \box_use_drop:N \l_@@_cell_box
1306 }
1307 { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1308 { \l_@@_pgf_node_code_tl }
1309 \str_if_empty:NF \l_@@_name_str
1310 {
1311   \pgfnodealias
1312     { \l_@@_name_str - \int_use:N \c@iRow - \int_use:N \c@jCol }
1313     { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1314 }
1315 \endpgfpicture
1316 }

```

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.

```

1317 \cs_new_protected:Npn \@@_patch_node_for_cell:n #1
1318 {
1319   \cs_new_protected:Npn \@@_patch_node_for_cell:
1320   {
1321     \hbox_set:Nn \l_@@_cell_box
1322     {
1323       \box_move_up:nn { \box_ht:N \l_@@_cell_box }
1324       \hbox_overlap_left:n
1325       {
1326         \pgfsys@markposition
1327         { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - NW }

```

I don't know why the following ajustement 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.

```

1328       #1
1329     }
1330     \box_use:N \l_@@_cell_box
1331     \box_move_down:nn { \box_dp:N \l_@@_cell_box }
1332     \hbox_overlap_left:n
1333     {
1334       \pgfsys@markposition
1335       { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - SE }
1336       #1
1337     }
1338   }
1339 }
1340 }

```

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

```

1341 \bool_lazy_or:nntf \sys_if_engine_xetex_p: \sys_if_output_dvi_p:
1342 {
1343   \@@_patch_node_for_cell:n
1344   { \skip_horizontal:n { 0.5 \box_wd:N \l_@@_cell_box } }
1345 }
1346 { \@@_patch_node_for_cell:n { } }

```

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

For example, for the following matrix,

```

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

```

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

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

```

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

```

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

```

1347 \cs_new_protected:Npn \@@_instruction_of_type:nnn #1 #2 #3
1348 {
1349   \bool_if:nTF { #1 } \tl_gput_left:cx \tl_gput_right:cx
1350   { \g_@@_#2 _ lines _ tl }
1351   {
1352     \use:c { @@ _ draw _ #2 : nnn }
1353     { \int_use:N \c@iRow }
1354     { \int_use:N \c@jCol }
1355     { \exp_not:n { #3 } }
1356   }
1357 }

1358 \cs_new_protected:Npn \@@_array:n
1359 {
1360   % modified 05-08-23
1361   \dim_set:Nn \col@sep
1362   { \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
1363   \dim_compare:nNnTF \l_@@_tabular_width_dim = \c_zero_dim
1364   { \cs_set_nopar:Npn \@halignto { } }
1365   { \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`.

```

1366   \@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.

```

1367   [ \str_if_eq:VnTF \l_@@_baseline_tl c c t ]
1368 }
1369 \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.

```

1370 \cs_set_eq:NN \@@_old_ialign: \ialign

```


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

```

1371 \cs_new_protected:Npn \@@_create_row_node:
1372 {
1373   \int_compare:nNnT \c@iRow > \g_@@_last_row_node_int
1374   {
1375     \int_gset_eq:NN \g_@@_last_row_node_int \c@iRow
1376     \@@_create_row_node_i:
1377   }
1378 }
1379 \cs_new_protected:Npn \@@_create_row_node_i:
1380 {

```

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

```

1381   \hbox
1382   {
1383     \bool_if:NT \l_@@_code_before_bool
1384     {
1385       \vtop
1386       {
1387         \skip_vertical:N 0.5\arrayrulewidth
1388         \pgfsys@markposition
1389         { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1390         \skip_vertical:N -0.5\arrayrulewidth
1391       }
1392     }
1393     \pgfpicture
1394     \pgfrememberpicturepositiononpagetrue
1395     \pgfcoordinate { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1396     { \pgfpoint \c_zero_dim { - 0.5 \arrayrulewidth } }
1397     \str_if_empty:NF \l_@@_name_str
1398     {
1399       \pgfnodealias
1400       { \l_@@_name_str - row - \int_eval:n { \c@iRow + 1 } }
1401       { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1402     }
1403     \endpgfpicture
1404   }
1405 }

```

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

```

1406 \cs_new:Npn \@@_everycr: { \noalign { \@@_everycr_i: } }
1407 \cs_new_protected:Npn \@@_everycr_i:
1408 {
1409   \int_gzero:N \c@jCol
1410   \bool_gset_false:N \g_@@_after_col_zero_bool
1411   \bool_if:NF \g_@@_row_of_col_done_bool
1412   {
1413     \@@_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).

```

1414   \tl_if_empty:NF \l_@@_hlines_clist
1415   {
1416     \tl_if_eq:NnF \l_@@_hlines_clist { all }
1417     {
1418       \exp_args:NNx
1419       \clist_if_in:NnT
1420       \l_@@_hlines_clist
1421       { \int_eval:n { \c@iRow + 1 } }
1422     }
1423   }

```

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.

```

1424         \int_compare:nNnT \c@iRow > { -1 }
1425         {
1426             \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@`.

```

1427             { \hrule height \arrayrulewidth width \c_zero_dim }
1428         }
1429     }
1430 }
1431 }
1432 }

```

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

```

1433 \cs_set_protected:Npn \@@_renew_dots:
1434 {
1435     \cs_set_eq:NN \ldots \@@_Ldots
1436     \cs_set_eq:NN \cdots \@@_Cdots
1437     \cs_set_eq:NN \vdots \@@_Vdots
1438     \cs_set_eq:NN \ddots \@@_Ddots
1439     \cs_set_eq:NN \iddots \@@_Iddots
1440     \cs_set_eq:NN \dots \@@_Ldots
1441     \cs_set_eq:NN \hdotsfor \@@_Hdotsfor:
1442 }

```

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

```

1443 \cs_new_protected:Npn \@@_colortbl_like:
1444 {
1445     \cs_set_eq:NN \cellcolor \@@_cellcolor_tabular
1446     \cs_set_eq:NN \rowcolor \@@_rowcolor_tabular
1447     \cs_set_eq:NN \columncolor \@@_columncolor_preamble
1448     \cs_set_eq:NN \rowcolors \@@_rowcolors_tabular
1449     \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors_tabular
1450 }

```

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

```

1451 \cs_new_protected:Npn \@@_pre_array_ii:
1452 {

```

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

```

1453     \int_gzero:N \g_@@_total_X_weight_int
1454     \@@_expand_clist:N \l_@@_hlines_clist
1455     \@@_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.

```

1456     \IfPackageLoadedTF { booktabs }
1457     { \tl_put_left:Nn \@BTnormal \@@_create_row_node_i: }
1458     { }

```

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

```

1459 \box_clear_new:N \l_@@_cell_box
1460 \normalbaselines

```

If the option `small` is used, we have to do some tuning. In particular, we change the value of `\arraystretch` (this parameter is used in the construction of `\@arstrutbox` in the beginning of `{array}`).

```

1461 \bool_if:NT \l_@@_small_bool
1462 {
1463     \cs_set_nopar:Npn \arraystretch { 0.47 }
1464     \dim_set:Nn \arraycolsep { 1.45 pt }
1465 }

1466 \bool_if:NT \g_@@_recreate_cell_nodes_bool
1467 {
1468     \tl_put_right:Nn \@@_begin_of_row:
1469     {
1470         \pgfsys@markposition
1471         { \@@_env: - row - \int_use:N \c@iRow - base }
1472     }
1473 }

```

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`.

```

1474 \cs_set_nopar:Npn \ialign
1475 {
1476     \IfPackageLoadedTF { colortbl }
1477     {
1478         \CT@everycr
1479         {
1480             \noalign { \cs_gset_eq:NN \CT@row@color \prg_do_nothing: }
1481             \@@_everycr:
1482         }
1483     }
1484     { \everycr { \@@_everycr: } }
1485     \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`.

```

1486 \dim_gzero_new:N \g_@@_dp_row_zero_dim
1487 \dim_gset:Nn \g_@@_dp_row_zero_dim { \box_dp:N \@arstrutbox }
1488 \dim_gzero_new:N \g_@@_ht_row_zero_dim
1489 \dim_gset:Nn \g_@@_ht_row_zero_dim { \box_ht:N \@arstrutbox }
1490 \dim_gzero_new:N \g_@@_ht_row_one_dim
1491 \dim_gset:Nn \g_@@_ht_row_one_dim { \box_ht:N \@arstrutbox }
1492 \dim_gzero_new:N \g_@@_dp_ante_last_row_dim
1493 \dim_gzero_new:N \g_@@_ht_last_row_dim
1494 \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1495 \dim_gzero_new:N \g_@@_dp_last_row_dim
1496 \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`.

```

1497 \cs_set_eq:NN \ialign \@@_old_ialign:

```

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

```

1498     \halign
1499 }

```

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).

```

1500 \cs_set_eq:NN \@@_old_ldots \ldots
1501 \cs_set_eq:NN \@@_old_cdots \cdots
1502 \cs_set_eq:NN \@@_old_vdots \vdots
1503 \cs_set_eq:NN \@@_old_ddots \ddots
1504 \cs_set_eq:NN \@@_old_iddots \iddots
1505 \bool_if:NTF \l_@@_standard_cline_bool
1506 { \cs_set_eq:NN \cline \@@_standard_cline }
1507 { \cs_set_eq:NN \cline \@@_cline }
1508 \cs_set_eq:NN \Ldots \@@_Ldots
1509 \cs_set_eq:NN \Cdots \@@_Cdots
1510 \cs_set_eq:NN \Vdots \@@_Vdots
1511 \cs_set_eq:NN \Ddots \@@_Ddots
1512 \cs_set_eq:NN \Iddots \@@_Iddots
1513 \cs_set_eq:NN \Hline \@@_Hline:
1514 \cs_set_eq:NN \Hspace \@@_Hspace:
1515 \cs_set_eq:NN \Hdotsfor \@@_Hdotsfor:
1516 \cs_set_eq:NN \Vdotsfor \@@_Vdotsfor:
1517 \cs_set_eq:NN \Block \@@_Block:
1518 \cs_set_eq:NN \rotate \@@_rotate:
1519 \cs_set_eq:NN \OnlyMainNiceMatrix \@@_OnlyMainNiceMatrix:n
1520 \cs_set_eq:NN \dotfill \@@_dotfill:
1521 \cs_set_eq:NN \CodeAfter \@@_CodeAfter:
1522 \cs_set_eq:NN \diagbox \@@_diagbox:nn
1523 \cs_set_eq:NN \NotEmpty \@@_NotEmpty:
1524 \cs_set_eq:NN \RowStyle \@@_RowStyle:n
1525 \seq_map_inline:Nn \l_@@_custom_line_commands_seq
1526 { \cs_set_eq:cc { ##1 } { nicematrix - ##1 } }
1527 \bool_if:NT \l_@@_color_inside_bool \@@_colortbl_like:
1528 \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.

```

1529 \cs_set_eq:NN \multicolumn \@@_multicolumn:nnn
1530 \hook_gput_code:nnn { env / tabular / begin } { . }
1531 { \cs_set_eq:NN \multicolumn \@@_old_multicolumn }

```

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

```

1532 \tl_if_exist:NT \l_@@_note_in_caption_tl
1533 {
1534   \tl_if_empty:NF \l_@@_note_in_caption_tl
1535   {
1536     \int_gset_eq:NN \g_@@_notes_caption_int \l_@@_note_in_caption_tl
1537     \int_gset:Nn \c@tabularnote { \l_@@_note_in_caption_tl }
1538   }
1539 }

```

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).

```

1540 \seq_gclear:N \g_@@_multicolumn_cells_seq
1541 \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).

```
1542 \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`).

```
1543 \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.

```
1544 \int_gzero_new:N \g_@@_col_total_int
```

```
1545 \cs_set_eq:NN \@ifnextchar \new@ifnextchar
```

```
1546 \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.

```
1547 \tl_gclear_new:N \g_@@_Cdots_lines_tl
1548 \tl_gclear_new:N \g_@@_Ldots_lines_tl
1549 \tl_gclear_new:N \g_@@_Vdots_lines_tl
1550 \tl_gclear_new:N \g_@@_Ddots_lines_tl
1551 \tl_gclear_new:N \g_@@_Iddots_lines_tl
1552 \tl_gclear_new:N \g_@@_HVdotsfor_lines_tl
```

```
1553 \tl_gclear:N \g_nicematrix_code_before_tl
```

```
1554 \tl_gclear:N \g_@@_pre_code_before_tl
```

```
1555 }
```

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

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

```
1556 \cs_new_protected:Npn \@@_pre_array:
```

```
1557 {
```

```
1558 \cs_if_exist:NT \theiRow { \int_set_eq:NN \l_@@_old_iRow_int \c@iRow }
```

```
1559 \int_gzero_new:N \c@iRow
```

```
1560 \cs_if_exist:NT \thejCol { \int_set_eq:NN \l_@@_old_jCol_int \c@jCol }
```

```
1561 \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).

```
1562 \int_compare:nNnT \l_@@_last_row_int = { -1 }
```

```
1563 {
```

```
1564 \bool_set_true:N \l_@@_last_row_without_value_bool
```

```
1565 \bool_if:NT \g_@@_aux_found_bool
```

```
1566 { \int_set:Nn \l_@@_last_row_int { \seq_item:Nn \g_@@_size_seq 3 } }
```

```
1567 }
```

```
1568 \int_compare:nNnT \l_@@_last_col_int = { -1 }
```

```
1569 {
```

```
1570 \bool_if:NT \g_@@_aux_found_bool
```

```
1571 { \int_set:Nn \l_@@_last_col_int { \seq_item:Nn \g_@@_size_seq 6 } }
```

```
1572 }
```

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”.

```

1573 \int_compare:nNnT \l_@@_last_row_int > { -2 }
1574 {
1575   \tl_put_right:Nn \@@_update_for_first_and_last_row:
1576   {
1577     \dim_gset:Nn \g_@@_ht_last_row_dim
1578     { \dim_max:nn \g_@@_ht_last_row_dim { \box_ht:N \l_@@_cell_box } }
1579     \dim_gset:Nn \g_@@_dp_last_row_dim
1580     { \dim_max:nn \g_@@_dp_last_row_dim { \box_dp:N \l_@@_cell_box } }
1581   }
1582 }

1583 \seq_gclear:N \g_@@_cols_vlism_seq
1584 \seq_gclear:N \g_@@_submatrix_seq

```

Now the `\CodeBefore`.

```

1585 \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.

```

1586 \seq_gclear:N \g_@@_pos_of_blocks_seq

```

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

```

1587 \seq_gclear_new:N \g_@@_multicolumn_cells_seq
1588 \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.

```

1589 \int_gset:Nn \g_@@_last_row_node_int { -2 }

```

The value `−2` is important.

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

```

1590 \@@_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.

```

1591 \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]`).

```

1592 \dim_zero_new:N \l_@@_left_delim_dim
1593 \dim_zero_new:N \l_@@_right_delim_dim
1594 \bool_if:NTF \g_@@_delims_bool
1595 {

```

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

```

1596   \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_left_delim_tl $ }
1597   \dim_set:Nn \l_@@_left_delim_dim { \box_wd:N \l_tmpa_box }
1598   \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_right_delim_tl $ }
1599   \dim_set:Nn \l_@@_right_delim_dim { \box_wd:N \l_tmpa_box }
1600 }
1601 {
1602   % modified 05-08-23
1603   \dim_gset:Nn \l_@@_left_delim_dim
1604   { 2 \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
1605   \dim_gset:Nn \l_@@_right_delim_dim \l_@@_left_delim_dim
1606 }

```

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).

```

1607 \hbox_set:Nw \l_@@_the_array_box
1608 \skip_horizontal:N \l_@@_left_margin_dim
1609 \skip_horizontal:N \l_@@_extra_left_margin_dim
1610 \c_math_toggle_token
1611 \bool_if:NTF \l_@@_light_syntax_bool
1612   { \use:c { @@-light-syntax } }
1613   { \use:c { @@-normal-syntax } }
1614 }

```

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

```

1615 \cs_new_protected_nopar:Npn \@@_CodeBefore_Body:w #1 \Body
1616 {
1617   \tl_gput_left:Nn \g_@@_pre_code_before_tl { #1 }
1618   \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...

```

1619 \@@_pre_array:
1620 }

```

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).

```

1621 \cs_new_protected:Npn \@@_pre_code_before:
1622 {

```

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).

```

1623 \int_set:Nn \c@iRow { \seq_item:Nn \g_@@_size_seq 2 }
1624 \int_set:Nn \c@jCol { \seq_item:Nn \g_@@_size_seq 5 }
1625 \int_set_eq:NN \g_@@_row_total_int { \seq_item:Nn \g_@@_size_seq 3 }
1626 \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.

```

1627 \pgfsys@markposition { \@@_env: - position }
1628 \pgfsys@getposition { \@@_env: - position } \@@_picture_position:
1629 \pgfpicture
1630 \pgf@relevantforpicturesizefalse

```

First, the recreation of the row nodes.

```

1631 \int_step_inline:nnn \l_@@_first_row_int { \g_@@_row_total_int + 1 }
1632 {
1633   \pgfsys@getposition { \@@_env: - row - ##1 } \@@_node_position:
1634   \pgfcoordinate { \@@_env: - row - ##1 }
1635   { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1636 }

```

Now, the recreation of the col nodes.

```

1637 \int_step_inline:nnn \l_@@_first_col_int { \g_@@_col_total_int + 1 }
1638 {
1639   \pgfsys@getposition { \@@_env: - col - ##1 } \@@_node_position:
1640   \pgfcoordinate { \@@_env: - col - ##1 }
1641   { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1642 }

```

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

```

1643 \@@_create_diag_nodes:

```

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

```

1644 \bool_if:NT \g_@@_recreate_cell_nodes_bool \@@_recreate_cell_nodes:
1645 \endpgfpicture

```

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

```

1646 \@@_create_blocks_nodes:
1647 \IfPackageLoadedTF { tikz }
1648 {
1649   \tikzset
1650   {
1651     every~picture / .style =
1652     { overlay , name~prefix = \@@_env: - }
1653   }
1654 }
1655 { }
1656 \cs_set_eq:NN \cellcolor \@@_cellcolor
1657 \cs_set_eq:NN \rectanglecolor \@@_rectanglecolor
1658 \cs_set_eq:NN \roundedrectanglecolor \@@_roundedrectanglecolor
1659 \cs_set_eq:NN \rowcolor \@@_rowcolor
1660 \cs_set_eq:NN \rowcolors \@@_rowcolors
1661 \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors
1662 \cs_set_eq:NN \arraycolor \@@_arraycolor
1663 \cs_set_eq:NN \columncolor \@@_columncolor
1664 \cs_set_eq:NN \chessboardcolors \@@_chessboardcolors
1665 \cs_set_eq:NN \SubMatrix \@@_SubMatrix_in_code_before
1666 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
1667 \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
1668 }

```

```

1669 \cs_new_protected:Npn \@@_exec_code_before:
1670 {
1671   \seq_gclear_new:N \g_@@_colors_seq
1672   \bool_gset_false:N \g_@@_recreate_cell_nodes_bool
1673   \group_begin:

```

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

```

1674 \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`).

```

1675 \int_compare:nNtT { \char_value_catcode:n { 60 } } = { 13 }
1676 {
1677   \@@_rescan_for_spanish:N \g_@@_pre_code_before_tl
1678   \@@_rescan_for_spanish:N \l_@@_code_before_tl
1679 }

```


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`.

```
1680 \exp_last_unbraced:NV \@@_CodeBefore_keys:
1681 \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.

```
1682 \@@_actually_color:
1683 \l_@@_code_before_tl
1684 \q_stop
1685 \bool_if:NT \l_@@_tabular_bool \c_math_toggle_token
1686 \group_end:
1687 \bool_if:NT \g_@@_recreate_cell_nodes_bool
1688 { \tl_put_left:Nn \@@_node_for_cell: \@@_patch_node_for_cell: }
1689 }
```

```
1690 \keys_define:nn { NiceMatrix / CodeBefore }
1691 {
1692   create-cell-nodes .bool_gset:N = \g_@@_recreate_cell_nodes_bool ,
1693   create-cell-nodes .default:n = true ,
1694   sub-matrix .code:n = \keys_set:nn { NiceMatrix / sub-matrix } { #1 } ,
1695   sub-matrix .value_required:n = true ,
1696   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1697   delimiters / color .value_required:n = true ,
1698   unknown .code:n = \@@_error:n { Unknown-key-for-CodeBefore }
1699 }
1700 \NewDocumentCommand \@@_CodeBefore_keys: { 0 { } }
1701 {
1702   \keys_set:nn { NiceMatrix / CodeBefore } { #1 }
1703   \@@_CodeBefore:w
1704 }
```

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.

```
1705 \cs_new_protected:Npn \@@_CodeBefore:w #1 \q_stop
1706 {
1707   \bool_if:NT \g_@@_aux_found_bool
1708   {
1709     \@@_pre_code_before:
1710     #1
1711   }
1712 }
```

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.

```
1713 \cs_new_protected:Npn \@@_recreate_cell_nodes:
1714 {
1715   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
1716   {
1717     \pgfsys@getposition { \@@_env: - ##1 - base } \@@_node_position:
1718     \pgfcoordinate { \@@_env: - row - ##1 - base }
1719     { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1720     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
1721     {
```

```

1722 \cs_if_exist:cT
1723 { pgf @ sys @ pdf @ mark @ pos @ \@@_env: - ##1 - #####1 - NW }
1724 {
1725     \pgfsys@getposition
1726     { \@@_env: - ##1 - #####1 - NW }
1727     \@@_node_position:
1728     \pgfsys@getposition
1729     { \@@_env: - ##1 - #####1 - SE }
1730     \@@_node_position_i:
1731     \@@_pgf_rect_node:nnn
1732     { \@@_env: - ##1 - #####1 }
1733     { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1734     { \pgfpointdiff \@@_picture_position: \@@_node_position_i: }
1735 }
1736 }
1737 }
1738 \int_step_inline:nn \c@iRow
1739 {
1740     \pgfnodealias
1741     { \@@_env: - ##1 - last }
1742     { \@@_env: - ##1 - \int_use:N \c@jCol }
1743 }
1744 \int_step_inline:nn \c@jCol
1745 {
1746     \pgfnodealias
1747     { \@@_env: - last - ##1 }
1748     { \@@_env: - \int_use:N \c@iRow - ##1 }
1749 }
1750 \@@_create_extra_nodes:
1751 }

1752 \cs_new_protected:Npn \@@_create_blocks_nodes:
1753 {
1754     \pgfpicture
1755     \pgf@relevantforpicturesizefalse
1756     \pgfrememberpicturepositiononpagetrue
1757     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
1758     { \@@_create_one_block_node:nnnnn ##1 }
1759     \endpgfpicture
1760 }

```

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

```

1761 \cs_new_protected:Npn \@@_create_one_block_node:nnnnn #1 #2 #3 #4 #5
1762 {
1763     \tl_if_empty:nF { #5 }
1764     {
1765         \@@_qpoint:n { col - #2 }
1766         \dim_set_eq:NN \l_tmpa_dim \pgf@x
1767         \@@_qpoint:n { #1 }
1768         \dim_set_eq:NN \l_tmpb_dim \pgf@y
1769         \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
1770         \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
1771         \@@_qpoint:n { \int_eval:n { #3 + 1 } }
1772         \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
1773         \@@_pgf_rect_node:nnnnn
1774         { \@@_env: - #5 }
1775         { \dim_use:N \l_tmpa_dim }
1776         { \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).

```

1777         { \dim_use:N \l_@@_tmpc_dim }
1778         { \dim_use:N \l_@@_tmpd_dim }
1779     }
1780 }

1781 \cs_new_protected:Npn \@@_patch_for_revtext:
1782 {
1783     \cs_set_eq:NN \@addamp \@addamp@LaTeX
1784     \cs_set_eq:NN \insert@column \insert@column@array
1785     \cs_set_eq:NN \@classx \@classx@array
1786     \cs_set_eq:NN \@xarraycr \@xarraycr@array
1787     \cs_set_eq:NN \@arraycr \@arraycr@array
1788     \cs_set_eq:NN \@xargarraycr \@xargarraycr@array
1789     \cs_set_eq:NN \array \array@array
1790     \cs_set_eq:NN \@array \@array@array
1791     \cs_set_eq:NN \@tabular \@tabular@array
1792     \cs_set_eq:NN \@mkpream \@mkpream@array
1793     \cs_set_eq:NN \endarray \endarray@array
1794     \cs_set:Npn \@tabarray { \@ifnextchar [ { \@array } { \@array [ c ] } }
1795     \cs_set:Npn \endtabular { \endarray $\egroup} % $
1796 }

```

11 The environment `{NiceArrayWithDelims}`

```

1797 \NewDocumentEnvironment { NiceArrayWithDelims }
1798 { m m O { } m ! O { } t \CodeBefore }
1799 {
1800     \bool_if:NT \c_@@_revtex_bool \@@_patch_for_revtext:
1801     \@@_provide_pgfsyspdfmark:
1802     \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.

```

1803     \bgroup

1804     \tl_gset:Nn \g_@@_left_delim_tl { #1 }
1805     \tl_gset:Nn \g_@@_right_delim_tl { #2 }
1806     \tl_gset:Nn \g_@@_user_preamble_tl { #4 }

1807     \int_gzero:N \g_@@_block_box_int
1808     \dim_zero:N \g_@@_width_last_col_dim
1809     \dim_zero:N \g_@@_width_first_col_dim
1810     \bool_gset_false:N \g_@@_row_of_col_done_bool
1811     \str_if_empty:NT \g_@@_name_env_str
1812     { \str_gset:Nn \g_@@_name_env_str { NiceArrayWithDelims } }
1813     \bool_if:NTF \l_@@_tabular_bool
1814     \mode_leave_vertical:
1815     \@@_test_if_math_mode:
1816     \bool_if:NT \l_@@_in_env_bool { \@@_fatal:n { Yet~in~env } }
1817     \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.

```
1818 \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.

```
1819 \cs_if_exist:NT \tikz@library@external@loaded
1820 {
1821   \tikzexternaldisable
1822   \cs_if_exist:NT \ifstandalone
1823   { \tikzset { external / optimize = false } }
1824 }
```

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

```
1825 \int_gincr:N \g_@@_env_int
1826 \bool_if:NF \l_@@_block_auto_columns_width_bool
1827 { \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`).

```
1828 \seq_gclear:N \g_@@_blocks_seq
1829 \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`.

```
1830 \seq_gclear:N \g_@@_pos_of_stroken_blocks_seq
1831 \seq_gclear:N \g_@@_pos_of_xdots_seq
1832 \tl_gclear_new:N \g_@@_code_before_tl
1833 \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.

```
1834 \tl_if_exist:cTF { c_@@ _ \int_use:N \g_@@_env_int _ tl }
1835 {
1836   \bool_gset_true:N \g_@@_aux_found_bool
1837   \use:c { c_@@ _ \int_use:N \g_@@_env_int _ tl }
1838 }
1839 { \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.

```
1840 \tl_gclear:N \g_@@_aux_tl
1841 \tl_if_empty:NF \g_@@_code_before_tl
1842 {
1843   \bool_set_true:N \l_@@_code_before_bool
1844   \tl_put_right:NV \l_@@_code_before_tl \g_@@_code_before_tl
1845 }
1846 \tl_if_empty:NF \g_@@_pre_code_before_tl
1847 { \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`.

```
1848 \bool_if:NTF \g_@@_delims_bool
1849 { \keys_set:nn { NiceMatrix / pNiceArray } }
1850 { \keys_set:nn { NiceMatrix / NiceArray } }
1851 { #3 , #5 }
```

```
1852 \@@_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:.`

```
1853   \IfBooleanTF { #6 } \@@_CodeBefore_Body:w \@@_pre_array:
1854 }
```

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

```
1855 {
1856   \bool_if:NTF \l_@@_light_syntax_bool
1857   { \use:c { end @@-light-syntax } }
1858   { \use:c { end @@-normal-syntax } }
1859   \c_math_toggle_token
1860   \skip_horizontal:N \l_@@_right_margin_dim
1861   \skip_horizontal:N \l_@@_extra_right_margin_dim
1862   \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.

```
1863   \bool_if:NT \l_@@_width_used_bool
1864   {
1865     \int_compare:nNnT \g_@@_total_X_weight_int = 0
1866     { \@@_error_or_warning:n { width-without~X-columns } }
1867   }
```

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 .

```
1868   \int_compare:nNnT \g_@@_total_X_weight_int > 0
1869   {
1870     \tl_gput_right:Nx \g_@@_aux_tl
1871     {
1872       \bool_set_true:N \l_@@_X_columns_aux_bool
1873       \dim_set:Nn \l_@@_X_columns_dim
1874       {
1875         \dim_compare:nNnTF
1876         {
1877           \dim_abs:n
1878           { \l_@@_width_dim - \box_wd:N \l_@@_the_array_box }
1879         }
1880         <
1881         { 0.001 pt }
1882         { \dim_use:N \l_@@_X_columns_dim }
1883         {
1884           \dim_eval:n
1885           {
1886             ( \l_@@_width_dim - \box_wd:N \l_@@_the_array_box )
1887             / \int_use:N \g_@@_total_X_weight_int
1888             + \l_@@_X_columns_dim
1889           }
1890         }
1891       }
1892     }
1893   }
```

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).

```
1894   \int_compare:nNnT \l_@@_last_row_int > { -2 }
1895   {
1896     \bool_if:NF \l_@@_last_row_without_value_bool
1897     {
1898       \int_compare:nNnF \l_@@_last_row_int = \c_iRow
1899       {
```

```

1900         \@@_error:n { Wrong~last~row }
1901         \int_gset_eq:NN \l_@@_last_row_int \c@iRow
1902     }
1903 }
1904 }

```

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

```

1905     \int_gset_eq:NN \c@jCol \g_@@_col_total_int
1906     \bool_if:nTF \g_@@_last_col_found_bool
1907     { \int_gdecr:N \c@jCol }
1908     {
1909         \int_compare:nNnT \l_@@_last_col_int > { -1 }
1910         { \@@_error:n { last~col~not~used } }
1911     }

```

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

```

1912     \int_gset_eq:NN \g_@@_row_total_int \c@iRow
1913     \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).

```

1914     \int_compare:nNnT \l_@@_first_col_int = 0
1915     {
1916         % \skip_horizontal:N \col@sep % 05-08-23
1917         \skip_horizontal:N \g_@@_width_first_col_dim
1918     }

```

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

```

1919     \bool_if:nTF { ! \g_@@_delims_bool }
1920     {
1921         \str_case:VnF \l_@@_baseline_tl
1922         {
1923             b \@@_use_arraybox_with_notes_b:
1924             c \@@_use_arraybox_with_notes_c:
1925         }
1926         \@@_use_arraybox_with_notes:
1927     }

```

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).

```

1928     {
1929         \int_compare:nNnTF \l_@@_first_row_int = 0
1930         {
1931             \dim_set_eq:NN \l_tmpa_dim \g_@@_dp_row_zero_dim
1932             \dim_add:Nn \l_tmpa_dim \g_@@_ht_row_zero_dim
1933         }
1934         { \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

```

1935     \int_compare:nNnTF \l_@@_last_row_int > { -2 }
1936     {
1937         \dim_set_eq:NN \l_tmpb_dim \g_@@_ht_last_row_dim
1938         \dim_add:Nn \l_tmpb_dim \g_@@_dp_last_row_dim
1939     }
1940     { \dim_zero:N \l_tmpb_dim }
1941     \hbox_set:Nn \l_tmpa_box
1942     {

```

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 user have not set the value with the option `last row` (and we are in the first compilation).

```

1943         \c_math_toggle_token
1944         \@@_color:V \l_@@_delimiters_color_tl
1945         \exp_after:wN \left \g_@@_left_delim_tl
1946         \vcenter
1947         {

```

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.

```

1948         \skip_vertical:n { -\l_tmpa_dim - \arrayrulewidth }
1949         \hbox
1950         {
1951             \bool_if:NTF \l_@@_tabular_bool
1952             { \skip_horizontal:N -\tabcolsep }
1953             { \skip_horizontal:N -\arraycolsep }
1954             \@@_use_arraybox_with_notes_c:
1955             \bool_if:NTF \l_@@_tabular_bool
1956             { \skip_horizontal:N -\tabcolsep }
1957             { \skip_horizontal:N -\arraycolsep }
1958         }

```

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

```

1959         \skip_vertical:n { -\l_tmpb_dim + \arrayrulewidth }
1960     }

```

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.

```

1961         \@@_color:V \l_@@_delimiters_color_tl
1962         \exp_after:wN \right \g_@@_right_delim_tl
1963         \c_math_toggle_token
1964     }

```

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.

```

1965         \bool_if:NTF \l_@@_delimiters_max_width_bool
1966         {
1967             \@@_put_box_in_flow_bis:nn
1968             \g_@@_left_delim_tl \g_@@_right_delim_tl
1969         }
1970         \@@_put_box_in_flow:
1971     }

```

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).

```

1972         \bool_if:NT \g_@@_last_col_found_bool
1973         {
1974             \skip_horizontal:N \g_@@_width_last_col_dim
1975             % \skip_horizontal:N \col@sep % 2023-08-05
1976         }
1977         \bool_if:NT \l_@@_preamble_bool
1978         {
1979             \int_compare:nNnT \c@jCol < \g_@@_static_num_of_col_int
1980             { \@@_warning_gredirect_none:n { columns~not~used } }
1981         }
1982         \@@_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.

```

1983         \egroup

```

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

```

1984         \iow_now:Nn \@mainaux { \ExplSyntaxOn }
1985         \iow_now:Nn \@mainaux { \char_set_catcode_space:n { 32 } }
1986         \iow_now:Nx \@mainaux
1987         {
1988             \tl_gset:cn { c_@@_ \int_use:N \g_@@_env_int _ tl }

```

```

1989         { \exp_not:V \g_@@_aux_tl }
1990     }
1991     \iow_now:Nn \@mainaux { \ExplSyntaxOff }

1992     \bool_if:NT \g_@@_footnote_bool \endsavenotes
1993 }

```

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

12 We construct the preamble of the array

The final user provides a preamble, but we must convert that preamble into a preamble that will be given to `{array}` (of the package `array`).

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

```

1994 \cs_new_protected:Npn \@@_transform_preamble:
1995 {
1996     \@@_transform_preamble_i:
1997     \@@_transform_preamble_ii:
1998 }
1999 \cs_new_protected:Npn \@@_transform_preamble_i:
2000 {
2001     \int_gzero:N \c@jCol
2002     \group_begin:

```

The sequence `\g_@@_cols_vlism_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 `vlism`).

```

2003     \seq_gclear:N \g_@@_cols_vlism_seq

```

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

```

2004     \bool_gset_false:N \g_tmpb_bool

```

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

```

2005     \tl_gclear_new:N \g_@@_pre_cell_tl

```

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

```

2006     \int_zero:N \l_tmpa_int
2007     \tl_gclear:N \g_@@_array_preamble_tl
2008     \tl_if_eq:NnTF \l_@@_vlines_clist { all }
2009     {
2010         \tl_gset:Nn \g_@@_array_preamble_tl
2011         { ! { \skip_horizontal:N \arrayrulewidth } }
2012     }
2013     {
2014         \clist_if_in:NnT \l_@@_vlines_clist 1
2015         {
2016             \tl_gset:Nn \g_@@_array_preamble_tl
2017             { ! { \skip_horizontal:N \arrayrulewidth } }
2018         }
2019     }

```

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

```

2020     \exp_last_unbraced:NV \@@_make_preamble:n \g_@@_user_preamble_tl \q_stop
2021     \int_gset_eq:NN \g_@@_static_num_of_col_int \c@jCol

```


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

```

2022   \bool_if:NT \l_@@_color_inside_bool
2023   {
2024     \regex_replace_all:NnN
2025       \c_@@_columncolor_regex
2026       { \c { @@_columncolor_preamble } }
2027     \g_@@_array_preamble_tl
2028   }

```

We are not sure that the following TeX group is still necessary.

```

2029   \group_end:
2030 }

```

```

2031 \cs_new_protected:Npn \@@_transform_preamble_ii:
2032 {

```

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

```

2033   \bool_lazy_or:nnT
2034     { ! \str_if_eq_p:Vn \g_@@_left_delim_tl { . } }
2035     { ! \str_if_eq_p:Vn \g_@@_right_delim_tl { . } }
2036     { \bool_gset_true:N \g_@@_delims_bool }

```

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

```

2037   \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).

```

2038   \int_compare:nNnTF \l_@@_first_col_int = 0
2039     { \tl_gput_left:NV \g_@@_array_preamble_tl \c_@@_preamble_first_col_tl }
2040     {
2041       \bool_lazy_all:nT
2042         {
2043           { \bool_not_p:n \g_@@_delims_bool }
2044           { \bool_not_p:n \l_@@_tabular_bool }
2045           { \tl_if_empty_p:N \l_@@_vlines_clist }
2046           { \bool_not_p:n \l_@@_exterior_arraycolsep_bool }
2047         }
2048         { \tl_gput_left:Nn \g_@@_array_preamble_tl { @ { } } }
2049     }
2050   \int_compare:nNnTF \l_@@_last_col_int > { -1 }
2051     { \tl_gput_right:NV \g_@@_array_preamble_tl \c_@@_preamble_last_col_tl }
2052     {
2053       \bool_lazy_all:nT
2054         {
2055           { \bool_not_p:n \g_@@_delims_bool }
2056           { \bool_not_p:n \l_@@_tabular_bool }
2057           { \tl_if_empty_p:N \l_@@_vlines_clist }
2058           { \bool_not_p:n \l_@@_exterior_arraycolsep_bool }
2059         }
2060         { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { } } }
2061     }

```

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`).

```

2062   \dim_compare:nNnT \l_@@_tabular_width_dim = \c_zero_dim
2063     {
2064       \tl_gput_right:Nn \g_@@_array_preamble_tl
2065         { > { \@@_error_too_much_cols: } 1 }
2066     }
2067 }

```

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

```

2068 \cs_new_protected:Npn \@@_make_preamble:n #1
2069 {
2070   \str_if_eq:nnF { #1 } { \q_stop }
2071   {
2072     \cs_if_exist:cTF { @@ \token_to_str:N #1 }
2073     { \use:c { @@ \token_to_str:N #1 } { #1 } }
2074     {
2075       \str_if_eq:nVTF { #1 } \l_@@_letter_vlism_tl
2076       {
2077         \seq_gput_right:Nx \g_@@_cols_vlism_seq
2078         { \int_eval:n { \c@jCol + 1 } }
2079         \tl_gput_right:Nx \g_@@_array_preamble_tl
2080         { \exp_not:N ! { \skip_horizontal:N \arrayrulewidth } }
2081         \@@_make_preamble:n
2082       }
2083     }
2084   }

```

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).

```

2083   {
2084     \keys_if_exist:nnTF { NiceMatrix / ColumnTypes } { #1 }
2085     {
2086       \keys_set:nn { NiceMatrix / ColumnTypes } { #1 }
2087       \@@_make_preamble:n
2088     }
2089     {
2090       \cs_if_exist:cTF { NC @ find @ #1 }
2091       {
2092         \tl_set_eq:Nc \l_tmpb_tl { NC @ rewrite @ #1 }
2093         \exp_last_unbraced:NV \@@_make_preamble:n \l_tmpb_tl
2094       }
2095       {
2096         \tl_if_eq:nnT { #1 } { S }
2097         { \@@_fatal:n { unknown~column~type~S } }
2098         { \@@_fatal:nn { unknown~column~type } { #1 } }
2099       }
2100     }
2101   }
2102 }
2103 }
2104 }

```

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

```

2105 \cs_new:cpn { @@ \string c } { \@@_make_preamble_i:n }
2106 \cs_new:cpn { @@ \string l } { \@@_make_preamble_i:n }
2107 \cs_new:cpn { @@ \string r } { \@@_make_preamble_i:n }
2108 \cs_new:cpn { @@ \string > } { \@@_make_preamble_xiv:nn }
2109 \cs_new:cpn { @@ \string ! } { \@@_make_preamble_ii:nn }
2110 \cs_new:cpn { @@ \string @ } { \@@_make_preamble_ii:nn }
2111 \cs_new:cpn { @@ \string | } { \@@_make_preamble_iii:n }
2112 \cs_new:cpn { @@ \string p } { \@@_make_preamble_iv:n }
2113 \cs_new:cpn { @@ \string b } { \@@_make_preamble_iv:n }
2114 \cs_new:cpn { @@ \string m } { \@@_make_preamble_iv:n }
2115 \cs_new:cpn { @@ \string V } { \@@_make_preamble_v:nn }
2116 \cs_new:cpn { @@ \string w } { \@@_make_preamble_vi:nnnn { } }

```

```

2117 \cs_new:cpn { @@ \string W } { \@@_make_preamble_vi:nnnn { \@@_special_W: } }
2118 \cs_new:cpn { @@ \string S } { \@@_make_preamble_vii:nn }
2119 \cs_new:cpn { @@ \string ( } { \@@_make_preamble_viii:nn }
2120 \cs_new:cpn { @@ \string [ } { \@@_make_preamble_viii:nn }
2121 \cs_new:cpn { @@ \string \{ } { \@@_make_preamble_viii:nn }
2122 \cs_new:cpn { @@ \string \left } { \@@_make_preamble_viii_ii:nn }
2123 \cs_new:cpn { @@ \string ) } { \@@_make_preamble_ix:nn }
2124 \cs_new:cpn { @@ \string ] } { \@@_make_preamble_ix:nn }
2125 \cs_new:cpn { @@ \string \} } { \@@_make_preamble_ix:nn }
2126 \cs_new:cpn { @@ \string \right } { \@@_make_preamble_ix_ii:nn }
2127 \cs_new:cpn { @@ \string X } { \@@_make_preamble_x:nn }
2128 \cs_new:cpn { @@ \string * } { \@@_make_preamble_xvi:nnn }
2129 \cs_new:cpn { @@ \string \NC@find } { \@@_make_preamble_xx:n }

```

The token `\NC@find` is at the head of the definition of the columns type done by `\newcolumnstype`. We want that token to be no-op here.

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

```

2130 \cs_new_protected:Npn \@@_make_preamble_i:n #1
2131 {
2132   \tl_gput_right:NV \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2133   \tl_gclear:N \g_@@_pre_cell_tl
2134   \tl_gput_right:Nn \g_@@_array_preamble_tl
2135   {
2136     > { \@@_cell_begin:w \str_set:Nn \l_@@_hpos_cell_str { #1 } }
2137     #1
2138     < \@@_cell_end:
2139   }

```

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

```

2140   \int_gincr:N \c@jCol
2141   \@@_make_preamble_xi:n
2142 }

```

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

```

2143 \cs_new_protected:Npn \@@_make_preamble_ii:nn #1 #2
2144 {
2145   \tl_gput_right:Nn \g_@@_array_preamble_tl { #1 { #2 } }
2146   \@@_make_preamble:n
2147 }

```

For `|`

```

2148 \cs_new_protected:Npn \@@_make_preamble_iii:n #1
2149 {

```

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

```

2150   \int_incr:N \l_tmpa_int
2151   \@@_make_preamble_iii_i:n
2152 }

```

```

2153 \cs_new_protected:Npn \@@_make_preamble_iii_i:n #1
2154 {
2155   \str_if_eq:nnTF { #1 } |
2156   { \@@_make_preamble_iii:n | }
2157   { \@@_make_preamble_iii_ii:nn { } #1 }
2158 }

```

```

2159 \cs_new_protected:Npn \@@_make_preamble_iii_ii:nn #1 #2
2160 {
2161   \str_if_eq:nnTF { #2 } [
2162   { \@@_make_preamble_iii_ii:nw { #1 } [ ] }
2163   { \@@_make_preamble_iii_iii:nn { #2 } { #1 } }
2164 }
2165 \cs_new_protected:Npn \@@_make_preamble_iii_ii:nw #1 [ #2 ]
2166 { \@@_make_preamble_iii_ii:nn { #1 , #2 } }

```

```

2167 \cs_new_protected:Npn \@@_make_preamble_iii_iii:nn #1 #2
2168 {
2169   \@@_compute_rule_width:n { multiplicity = \l_tmpa_int , #2 }
2170   \tl_gput_right:Nx \g_@@_array_preamble_tl
2171   {

```

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

```

2172     \exp_not:N ! { \skip_horizontal:n { \dim_eval:n { \l_@@_rule_width_dim } } }
2173   }
2174   \tl_gput_right:Nx \g_@@_pre_code_after_tl
2175   {
2176     \@@_vline:n
2177     {
2178       position = \int_eval:n { \c@jCol + 1 } ,
2179       multiplicity = \int_use:N \l_tmpa_int ,
2180       total-width = \dim_use:N \l_@@_rule_width_dim ,
2181       #2
2182     }

```

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.

```

2183   }
2184   \int_zero:N \l_tmpa_int
2185   \str_if_eq:nnT { #1 } { \q_stop } { \bool_gset_true:N \g_tmpb_bool }
2186   \@@_make_preamble:n #1
2187 }
2188 \cs_new_protected:Npn \@@_make_preamble_xiv:nn #1 #2
2189 {
2190   \tl_gput_right:Nn \g_@@_pre_cell_tl { > { #2 } }
2191   \@@_make_preamble:n
2192 }
2193 \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.

```

2194 \keys_define:nn { WithArrows / p-column }
2195 {
2196   r .code:n = \str_set:Nn \l_@@_hpos_col_str { r } ,
2197   r .value_forbidden:n = true ,
2198   c .code:n = \str_set:Nn \l_@@_hpos_col_str { c } ,
2199   c .value_forbidden:n = true ,
2200   l .code:n = \str_set:Nn \l_@@_hpos_col_str { l } ,
2201   l .value_forbidden:n = true ,
2202   R .code:n =
2203     \IfPackageLoadedTF { ragged2e }
2204     { \str_set:Nn \l_@@_hpos_col_str { R } }
2205     {
2206       \@@_error_or_warning:n { ragged2e-not-loaded }
2207       \str_set:Nn \l_@@_hpos_col_str { r }
2208     } ,
2209   R .value_forbidden:n = true ,
2210   L .code:n =
2211     \IfPackageLoadedTF { ragged2e }
2212     { \str_set:Nn \l_@@_hpos_col_str { L } }
2213     {
2214       \@@_error_or_warning:n { ragged2e-not-loaded }
2215       \str_set:Nn \l_@@_hpos_col_str { l }
2216     } ,
2217   L .value_forbidden:n = true ,
2218   C .code:n =
2219     \IfPackageLoadedTF { ragged2e }
2220     { \str_set:Nn \l_@@_hpos_col_str { C } }

```

```

2221     {
2222         \@@_error_or_warning:n { ragged2e-not-loaded }
2223         \str_set:Nn \l_@@_hpos_col_str { c }
2224     } ,
2225     C .value_forbidden:n = true ,
2226     S .code:n = \str_set:Nn \l_@@_hpos_col_str { si } ,
2227     S .value_forbidden:n = true ,
2228     p .code:n = \str_set:Nn \l_@@_vpos_col_str { p } ,
2229     p .value_forbidden:n = true ,
2230     t .meta:n = p ,
2231     m .code:n = \str_set:Nn \l_@@_vpos_col_str { m } ,
2232     m .value_forbidden:n = true ,
2233     b .code:n = \str_set:Nn \l_@@_vpos_col_str { b } ,
2234     b .value_forbidden:n = true ,
2235 }

```

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

```

2236 \cs_new_protected:Npn \@@_make_preamble_iv:n #1
2237 {
2238     \str_set:Nn \l_@@_vpos_col_str { #1 }

```

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

```

2239     \@@_make_preamble_iv_i:n
2240 }

2241 \cs_new_protected:Npn \@@_make_preamble_iv_i:n #1
2242 {
2243     \str_if_eq:nnTF { #1 } { [ ]
2244         { \@@_make_preamble_iv_ii:w [ ] }
2245         { \@@_make_preamble_iv_ii:w [ ] { #1 } }
2246     }

2247     \cs_new_protected:Npn \@@_make_preamble_iv_ii:w [ #1 ]
2248     { \@@_make_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.

```

2249 \cs_new_protected:Npn \@@_make_preamble_iv_iii:nn #1 #2
2250 {

```

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).

```

2251     \str_set:Nn \l_@@_hpos_col_str { j }
2252     \tl_set:Nn \l_tmpa_tl { #1 }
2253     \@@_keys_p_column:V \l_tmpa_tl
2254     \@@_make_preamble_iv_iv:nn { #2 } { minipage }
2255 }

2256 \cs_new_protected:Npn \@@_keys_p_column:n #1
2257 { \keys_set_known:nnN { WithArrows / p-column } { #1 } \l_tmpa_tl }
2258 \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`.

```

2259 \cs_new_protected:Npn \@@_make_preamble_iv_iv:nn #1 #2
2260 {
2261     \use:x
2262     {
2263         \@@_make_preamble_iv_v:nnnnnnnn
2264         { \str_if_eq:VnTF \l_@@_vpos_col_str { p } { t } { b } }
2265         { \dim_eval:n { #1 } }
2266         {

```

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.

```

2267 \str_if_eq:VnTF \l_@@_hpos_col_str j
2268 { \str_set:Nn \exp_not:N \l_@@_hpos_cell_str { } }
2269 {
2270 \str_set:Nn \exp_not:N \l_@@_hpos_cell_str
2271 { \str_lowercase:V \l_@@_hpos_col_str }
2272 }
2273 \str_case:Vn \l_@@_hpos_col_str
2274 {
2275 c { \exp_not:N \centering }
2276 l { \exp_not:N \raggedright }
2277 r { \exp_not:N \raggedleft }
2278 C { \exp_not:N \Centering }
2279 L { \exp_not:N \RaggedRight }
2280 R { \exp_not:N \RaggedLeft }
2281 }
2282 }
2283 { \str_if_eq:VnT \l_@@_vpos_col_str { m } \@@_center_cell_box: }
2284 { \str_if_eq:VnT \l_@@_hpos_col_str { si } \siunitx_cell_begin:w }
2285 { \str_if_eq:VnT \l_@@_hpos_col_str { si } \siunitx_cell_end: }
2286 { #2 }
2287 {
2288 \str_case:VnF \l_@@_hpos_col_str
2289 {
2290 { j } { c }
2291 { si } { c }
2292 }

```

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

```

2293 { \str_lowercase:V \l_@@_hpos_col_str }
2294 }
2295 }

```

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

```

2296 \int_gincr:N \c@jCol
2297 \@@_make_preamble_xi:n
2298 }

```

#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*.

```

2299 \cs_new_protected:Npn \@@_make_preamble_iv_v:nnnnnnn #1 #2 #3 #4 #5 #6 #7 #8
2300 {
2301 \str_if_eq:VnTF \l_@@_hpos_col_str { si }
2302 { \tl_gput_right:Nn \g_@@_array_preamble_tl { > { \@@_test_if_empty_for_S: } } }
2303 { \tl_gput_right:Nn \g_@@_array_preamble_tl { > { \@@_test_if_empty: } } }
2304 \tl_gput_right:NV \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2305 \tl_gclear:N \g_@@_pre_cell_tl
2306 \tl_gput_right:Nn \g_@@_array_preamble_tl
2307 {

```

2308 > {

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.

```
2309 \dim_set:Nn \l_@@_col_width_dim { #2 }
2310 \@@_cell_begin:w
2311 \begin { #7 } [ #1 ] { #2 }
```

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

```
2312 \everypar
2313 {
2314 \vrule height \box_ht:N \@arstrutbox width \c_zero_dim
2315 \everypar { }
2316 }
```

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

2317 #3

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

```
2318 \g_@@_row_style_tl
2319 \arraybackslash
2320 #5
2321 }
2322 #8
2323 < {
2324 #6
```

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

```
2325 \@finalstrut \@arstrutbox
2326 % \bool_if:NT \g_@@_rotate_bool { \raggedright \hspace = 3 cm }
2327 \end { #7 }
```

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

```
2328 #4
2329 \@@_cell_end:
2330 }
2331 }
2332 }
```

```
2333 \cs_new_protected:Npn \@@_test_if_empty: \ignorespaces #1
2334 {
2335 \peek_meaning:NT \unskip
2336 {
2337 \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2338 {
2339 \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.

```
2340 \skip_horizontal:N \l_@@_col_width_dim
2341 }
2342 }
2343 #1
2344 }

2345 \cs_new_protected:Npn \@@_test_if_empty_for_S: #1
2346 {
2347 \peek_meaning:NT \__siunitx_table_skip:n
2348 {
2349 \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2350 { \box_set_wd:Nn \l_@@_cell_box \c_zero_dim }
2351 }
2352 #1
2353 }
```

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 than the height of `\@arstrutbox`, there is only one row.

```
2354 \cs_new_protected:Npn \@_center_cell_box:
2355 {
```

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

```
2356 \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2357 {
2358   \int_compare:nNnT
2359     { \box_ht:N \l_@@_cell_box }
2360     >
```

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).

```
2361     { \box_ht:N \strutbox }
2362     {
2363       \hbox_set:Nn \l_@@_cell_box
2364       {
2365         \box_move_down:nn
2366         {
2367           ( \box_ht:N \l_@@_cell_box - \box_ht:N \@arstrutbox
2368             + \baselineskip ) / 2
2369         }
2370         { \box_use:N \l_@@_cell_box }
2371       }
2372     }
2373   }
2374 }
```

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

```
2375 \cs_new_protected:Npn \@_make_preamble_v:nn #1 #2
2376 {
2377   \str_if_eq:nnTF { #2 } { [ ] }
2378   { \@_make_preamble_v_i:w [ ] }
2379   { \@_make_preamble_v_i:w [ ] { #2 } }
2380 }
2381 \cs_new_protected:Npn \@_make_preamble_v_i:w [ #1 ]
2382 { \@_make_preamble_v_ii:nn { #1 } }
2383 \cs_new_protected:Npn \@_make_preamble_v_ii:nn #1 #2
2384 {
2385   \str_set:Nn \l_@@_vpos_col_str { p }
2386   \str_set:Nn \l_@@_hpos_col_str { j }
2387   \tl_set:Nn \l_tmpa_tl { #1 }
2388   \@_keys_p_column:V \l_tmpa_tl
2389   \IfPackageLoadedTF { varwidth }
2390   { \@_make_preamble_iv_iv:nn { #2 } { varwidth } }
2391   {
2392     \@_error_or_warning:n { varwidth-not-loaded }
2393     \@_make_preamble_iv_iv:nn { #2 } { minipage }
2394   }
2395 }
```

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.

```
2396 \cs_new_protected:Npn \@_make_preamble_vi:nnnn #1 #2 #3 #4
```



```

2397 {
2398   \str_if_eq:nnTF { #3 } { s }
2399   { \@@_make_preamble_vi_i:nnnn { #1 } { #4 } }
2400   { \@@_make_preamble_vi_ii:nnnn { #1 } { #2 } { #3 } { #4 } }
2401 }

```

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.

```

2402 \cs_new_protected:Npn \@@_make_preamble_vi_i:nnnn #1 #2
2403 {
2404   \tl_gput_right:NV \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2405   \tl_gclear:N \g_@@_pre_cell_tl
2406   \tl_gput_right:Nn \g_@@_array_preamble_tl
2407   {
2408     > {
2409       \dim_set:Nn \l_@@_col_width_dim { #2 }
2410       \@@_cell_begin:w
2411       \str_set:Nn \l_@@_hpos_cell_str { c }
2412     }
2413     c
2414     < {
2415       \@@_cell_end_for_ws:
2416       #1
2417       \@@_adjust_size_box:
2418       \box_use_drop:N \l_@@_cell_box
2419     }
2420   }
2421   \int_gincr:N \c@jCol
2422   \@@_make_preamble_xi:n
2423 }

```

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

```

2424 \cs_new_protected:Npn \@@_make_preamble_vi_ii:nnnn #1 #2 #3 #4
2425 {
2426   \tl_gput_right:NV \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2427   \tl_gclear:N \g_@@_pre_cell_tl
2428   \tl_gput_right:Nn \g_@@_array_preamble_tl
2429   {
2430     > {

```

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.

```

2431       \dim_set:Nn \l_@@_col_width_dim { #4 }
2432       \hbox_set:Nw \l_@@_cell_box
2433       \@@_cell_begin:w
2434       \str_set:Nn \l_@@_hpos_cell_str { #3 }
2435     }
2436     c
2437     < {
2438       \@@_cell_end:
2439       \hbox_set_end:
2440       #1
2441       \@@_adjust_size_box:
2442       \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2443     }
2444   }

```

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

```

2445   \int_gincr:N \c@jCol
2446   \@@_make_preamble_xi:n
2447 }

```

```

2448 \cs_new_protected:Npn \@@_special_W:
2449 {
2450   \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \l_@@_col_width_dim
2451   { \@@_warning:n { W~warning } }
2452 }

```

For S (of siunitx).

```

2453 \cs_new_protected:Npn \@@_make_preamble_vii:nn #1 #2
2454 {
2455   \str_if_eq:nnTF { #2 } { [ ]
2456     { \@@_make_preamble_vii_i:w [ ]
2457       { \@@_make_preamble_vii_i:w [ ] { #2 } }
2458     }
2459   \cs_new_protected:Npn \@@_make_preamble_vii_i:w [ #1 ]
2460     { \@@_make_preamble_vii_ii:n { #1 } }
2461   \cs_new_protected:Npn \@@_make_preamble_vii_ii:n #1
2462   {
2463     \IfPackageAtLeastTF { siunitx } { 2022/01/01 }
2464     {
2465       \tl_gput_right:NV \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2466       \tl_gclear:N \g_@@_pre_cell_tl
2467       \tl_gput_right:Nn \g_@@_array_preamble_tl
2468       {
2469         > {
2470           \@@_cell_begin:w
2471           \keys_set:nn { siunitx } { #1 }
2472           \siunitx_cell_begin:w
2473         }
2474         c
2475         < { \siunitx_cell_end: \@@_cell_end: }
2476       }

```

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

```

2477   \int_gincr:N \c@jCol
2478   \@@_make_preamble_xi:n
2479 }
2480 { \@@_fatal:n { Version~of~siunitx~too~old } }
2481 }

```

For (, [and \{.

```

2482 \cs_new_protected:Npn \@@_make_preamble_viii:nn #1 #2
2483 {
2484   \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.

```

2485   \int_compare:nNnTF \c@jCol = \c_zero_int
2486   {
2487     \str_if_eq:VnTF \g_@@_left_delim_tl { . }
2488     {

```

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

```

2489       \tl_gset:Nn \g_@@_left_delim_tl { #1 }
2490       \tl_gset:Nn \g_@@_right_delim_tl { . }
2491       \@@_make_preamble:n #2
2492     }
2493   {
2494     \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2495     \@@_make_preamble_viii_i:nn { #1 } { #2 }
2496   }
2497 }
2498 { \@@_make_preamble_viii_i:nn { #1 } { #2 } }
2499 }

```

```

2500 \cs_new_protected:Npn \@@_make_preamble_viii_i:nn #1 #2
2501 {
2502   \tl_gput_right:Nx \g_@@_pre_code_after_tl
2503   { \@@_delimiter:nnn #1 { \int_eval:n { \c@jCol + 1 } } \c_true_bool }
2504   \tl_if_in:nnTF { ( [ \{ ) ] \} \left \right } { #2 }
2505   {
2506     \@@_error:nn { delimiter~after~opening } { #2 }
2507     \@@_make_preamble:n
2508   }
2509   { \@@_make_preamble:n #2 }
2510 }
2511 \cs_new_protected:Npn \@@_make_preamble_viii_ii:n #1
2512 { \@@_make_preamble_viii:nn }

```

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}`).

```

2513 \cs_new_protected:Npn \@@_make_preamble_ix:nn #1 #2
2514 {
2515   \bool_if:NT \l_@@_small_bool { \@@_fatal:n { Delimiter~with~small } }
2516   \tl_if_in:nnTF { ) ] \} } { #2 }
2517   { \@@_make_preamble_ix_i:nnn #1 #2 }
2518   {
2519     \tl_if_eq:nnTF { \q_stop } { #2 }
2520     {
2521       \str_if_eq:VnTF \g_@@_right_delim_tl { . }
2522       { \tl_gset:Nn \g_@@_right_delim_tl { #1 } }
2523       {
2524         \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2525         \tl_gput_right:Nx \g_@@_pre_code_after_tl
2526         { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2527         \@@_make_preamble:n #2
2528       }
2529     }
2530     {
2531       \tl_if_in:nnT { ( [ \{ \left } { #2 }
2532       { \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } } }
2533       \tl_gput_right:Nx \g_@@_pre_code_after_tl
2534       { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2535       \@@_make_preamble:n #2
2536     }
2537   }
2538 }
2539 \cs_new_protected:Npn \@@_make_preamble_ix_i:nnn #1 #2 #3
2540 {
2541   \tl_if_eq:nnTF { \q_stop } { #3 }
2542   {
2543     \str_if_eq:VnTF \g_@@_right_delim_tl { . }
2544     {
2545       \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2546       \tl_gput_right:Nx \g_@@_pre_code_after_tl
2547       { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2548       \tl_gset:Nn \g_@@_right_delim_tl { #2 }
2549     }
2550     {
2551       \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2552       \tl_gput_right:Nx \g_@@_pre_code_after_tl
2553       { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2554       \@@_error:nn { double~closing~delimiter } { #2 }
2555     }
2556   }

```

```

2557 {
2558   \tl_gput_right:Nx \g_@@_pre_code_after_tl
2559   { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2560   \@@_error:nn { double~closing-delimiter } { #2 }
2561   \@@_make_preamble:n #3
2562 }
2563 }
2564 \cs_new_protected:Npn \@@_make_preamble_ix_ii:n #1
2565 { \@@_make_preamble_ix:nn }

```

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.

```

2566 \cs_new_protected:Npn \@@_make_preamble_x:nn #1 #2
2567 {
2568   \str_if_eq:nnTF { #2 } { [ ]
2569     { \@@_make_preamble_x_i:w [ ]
2570       { \@@_make_preamble_x_i:w [ ] #2 }
2571     }
2572   \cs_new_protected:Npn \@@_make_preamble_x_i:w [ #1 ]
2573   { \@@_make_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).

```

2574 \keys_define:nn { WithArrows / X-column }
2575 { 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.

```

2576 \cs_new_protected:Npn \@@_make_preamble_x_ii:n #1
2577 {

```

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).

```

2578   \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).

```

2579   \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.

```

2580   \int_zero_new:N \l_@@_weight_int
2581   \int_set:Nn \l_@@_weight_int { 1 }
2582   \tl_set:Nn \l_tmpa_tl { #1 }
2583   \@@_keys_p_column:V \l_tmpa_tl
2584   \keys_set:nV { WithArrows / X-column } \l_tmpa_tl
2585   \int_compare:nNnT \l_@@_weight_int < 0
2586   {
2587     \@@_error_or_warning:n { negative~weight }
2588     \int_set:Nn \l_@@_weight_int { - \l_@@_weight_int }
2589   }
2590   \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).

```

2591   \bool_if:NTF \l_@@_X_columns_aux_bool
2592   {
2593     \exp_args:Nnx
2594     \@@_make_preamble_iv_iv:nn

```

```

2595     { \l_@@_weight_int \l_@@_X_columns_dim }
2596     { minipage }
2597   }
2598   {
2599     \tl_gput_right:Nn \g_@@_array_preamble_tl
2600     {
2601       > {
2602         \@@_cell_begin:w
2603         \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`.

```

2604         \NotEmpty

```

The following code will nullify the box of the cell.

```

2605         \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2606         { \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`.

```

2607         \begin { minipage } { 5 cm } \arraybackslash
2608       }
2609       c
2610       < {
2611         \end { minipage }
2612         \@@_cell_end:
2613       }
2614     }
2615     \int_gincr:N \c@jCol
2616     \@@_make_preamble_xi:n
2617   }
2618 }

```

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 `@{...}`.

```

2619 \cs_new_protected:Npn \@@_make_preamble_xi:n #1
2620 {
2621   \str_if_eq:nnTF { #1 } { < }
2622   \@@_make_preamble_xiii:n
2623   {
2624     \str_if_eq:nnTF { #1 } { @ }
2625     \@@_make_preamble_xv:n
2626     {
2627       \tl_if_eq:NnTF \l_@@_vlines_clist { all }
2628       {
2629         \tl_gput_right:Nn \g_@@_array_preamble_tl
2630         { ! { \skip_horizontal:N \arrayrulewidth } }
2631       }
2632       {
2633         \exp_args:NNx
2634         \clist_if_in:NnT \l_@@_vlines_clist { \int_eval:n { \c@jCol + 1 } }
2635         {
2636           \tl_gput_right:Nn \g_@@_array_preamble_tl
2637           { ! { \skip_horizontal:N \arrayrulewidth } }
2638         }
2639       }
2640       \@@_make_preamble:n { #1 }
2641     }
2642   }
2643 }

```

```

2644 \cs_new_protected:Npn \@@_make_preamble_xiii:n #1
2645 {
2646   \tl_gput_right:Nn \g_@@_array_preamble_tl { < { #1 } }
2647   \@@_make_preamble_xi:n
2648 }

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.

2649 \cs_new_protected:Npn \@@_make_preamble_xv:n #1
2650 {
2651   \tl_if_eq:NnTF \l_@@_vlines_clist { all }
2652   {
2653     \tl_gput_right:Nn \g_@@_array_preamble_tl
2654     { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2655   }
2656   {
2657     \exp_args:NNx
2658     \clist_if_in:NnTF \l_@@_vlines_clist { \int_eval:n { \c@jCol + 1 } }
2659     {
2660       \tl_gput_right:Nn \g_@@_array_preamble_tl
2661       { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2662     }
2663     { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { #1 } } }
2664   }
2665   \@@_make_preamble:n
2666 }

2667 \cs_new_protected:Npn \@@_make_preamble_xx:n #1 { \@@_make_preamble:n }

2668 \cs_new_protected:Npn \@@_make_preamble_xvi:nnn #1 #2 #3
2669 {
2670   \tl_clear:N \l_tmpa_tl
2671   \int_step_inline:nn { #2 } { \tl_put_right:Nn \l_tmpa_tl { #3 } }
2672   \exp_last_unbraced:NV \@@_make_preamble:n \l_tmpa_tl
2673 }

```

13 The redefinition of \multicolumn

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

```

2674 \cs_new:Npn \@@_multicolumn:nnn #1 #2 #3
2675 {

```

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.

```

2676   \multispan { #1 }
2677   \begingroup
2678   \cs_set:Npn \@addamp { \if@firstamp \@firstampfalse \else \@preamerr 5 \fi }

```

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

```

2679   \tl_gclear:N \g_@@_preamble_tl
2680   \@@_make_m_preamble:n #2 \q_stop

```

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

```

2681   \exp_args:NV \mkpream \g_@@_preamble_tl
2682   \@addtopreamble \@empty
2683   \endgroup

```

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

```

2684 \int_compare:nNnT { #1 } > 1
2685 {
2686   \seq_gput_left:Nx \g_@@_multicolumn_cells_seq
2687   { \int_use:N \c@iRow - \int_eval:n { \c@jCol + 1 } }
2688   \seq_gput_left:Nn \g_@@_multicolumn_sizes_seq { #1 }
2689   \seq_gput_right:Nx \g_@@_pos_of_blocks_seq
2690   {
2691     {
2692       \int_compare:nNnTF \c@jCol = 0
2693       { \int_eval:n { \c@iRow + 1 } }
2694       { \int_use:N \c@iRow }
2695     }
2696     { \int_eval:n { \c@jCol + 1 } }
2697     {
2698       \int_compare:nNnTF \c@jCol = 0
2699       { \int_eval:n { \c@iRow + 1 } }
2700       { \int_use:N \c@iRow }
2701     }
2702     { \int_eval:n { \c@jCol + #1 } }
2703     { } % for the name of the block
2704   }
2705 }

```

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

```

2706 \cs_set:Npn \@sharp { #3 }
2707 \@arstrut
2708 \@preamble
2709 \null

```

We add some lines.

```

2710 \int_gadd:Nn \c@jCol { #1 - 1 }
2711 \int_compare:nNnT \c@jCol > \g_@@_col_total_int
2712 { \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
2713 \ignorespaces
2714 }

```

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`.

```

2715 \cs_new_protected:Npn \@@_make_m_preamble:n #1
2716 {
2717   \str_case:nnF { #1 }
2718   {
2719     c { \@@_make_m_preamble_i:n #1 }
2720     l { \@@_make_m_preamble_i:n #1 }
2721     r { \@@_make_m_preamble_i:n #1 }
2722     > { \@@_make_m_preamble_ii:nn #1 }
2723     ! { \@@_make_m_preamble_ii:nn #1 }
2724     @ { \@@_make_m_preamble_ii:nn #1 }
2725     | { \@@_make_m_preamble_iii:n #1 }
2726     p { \@@_make_m_preamble_iv:nnn t #1 }
2727     m { \@@_make_m_preamble_iv:nnn c #1 }
2728     b { \@@_make_m_preamble_iv:nnn b #1 }
2729     w { \@@_make_m_preamble_v:nnnn { } #1 }
2730     W { \@@_make_m_preamble_v:nnnn { \@@_special_W: } #1 }
2731     \q_stop { }

```

The token `\NC@find` is at the head of the definition of the columns type done by `\newcolumnstype`. We wan't that token to no-op here.

```

2732 \NC@find { \@@_make_preamble:n }
2733 }

```

```

2734 {
2735   \cs_if_exist:cTF { NC @ find @ #1 }
2736   {
2737     \tl_set_eq:Nc \l_tmpa_tl { NC @ rewrite @ #1 }
2738     \exp_last_unbraced:NV \@@_make_m_preamble:n \l_tmpa_tl
2739   }
2740   {
2741     \tl_if_eq:nnT { #1 } { S }
2742     { \@@_fatal:n { unknown~column~type~S } }
2743     { \@@_fatal:nn { unknown~column~type } { #1 } }
2744   }
2745 }
2746 }

```

For c, l and r

```

2747 \cs_new_protected:Npn \@@_make_m_preamble_i:n #1
2748 {
2749   \tl_gput_right:Nn \g_@@_preamble_tl
2750   {
2751     > { \@@_cell_begin:w \str_set:Nn \l_@@_hpos_cell_str { #1 } }
2752     #1
2753     < \@@_cell_end:
2754   }

```

We test for the presence of a <.

```

2755   \@@_make_m_preamble_x:n
2756 }

```

For >, ! and @

```

2757 \cs_new_protected:Npn \@@_make_m_preamble_ii:nn #1 #2
2758 {
2759   \tl_gput_right:Nn \g_@@_preamble_tl { #1 { #2 } }
2760   \@@_make_m_preamble:n
2761 }

```

For |

```

2762 \cs_new_protected:Npn \@@_make_m_preamble_iii:n #1
2763 {
2764   \tl_gput_right:Nn \g_@@_preamble_tl { #1 }
2765   \@@_make_m_preamble:n
2766 }

```

For p, m and b

```

2767 \cs_new_protected:Npn \@@_make_m_preamble_iv:nnn #1 #2 #3
2768 {
2769   \tl_gput_right:Nn \g_@@_preamble_tl
2770   {
2771     > {
2772       \@@_cell_begin:w
2773       \begin { minipage } [ #1 ] { \dim_eval:n { #3 } }
2774       \mode_leave_vertical:
2775       \arraybackslash
2776       \vrule height \box_ht:N \@arstrutbox depth 0 pt width 0 pt
2777     }
2778     c
2779     < {
2780       \vrule height 0 pt depth \box_dp:N \@arstrutbox width 0 pt
2781       \end { minipage }
2782       \@@_cell_end:
2783     }
2784   }

```

We test for the presence of a <.

```

2785   \@@_make_m_preamble_x:n
2786 }

```


For w and W

```

2787 \cs_new_protected:Npn \@@_make_m_preamble_v:nnnn #1 #2 #3 #4
2788 {
2789   \tl_gput_right:Nn \g_@@_preamble_tl
2790   {
2791     > {
2792       \dim_set:Nn \l_@@_col_width_dim { #4 }
2793       \hbox_set:Nw \l_@@_cell_box
2794       \@@_cell_begin:w
2795       \str_set:Nn \l_@@_hpos_cell_str { #3 }
2796     }
2797     c
2798     < {
2799       \@@_cell_end:
2800       \hbox_set_end:
2801       \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
2802       #1
2803       \@@_adjust_size_box:
2804       \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2805     }
2806   }

```

We test for the presence of a <.

```

2807   \@@_make_m_preamble_x:n
2808 }

```

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

```

2809 \cs_new_protected:Npn \@@_make_m_preamble_x:n #1
2810 {
2811   \str_if_eq:nnTF { #1 } { < }
2812   \@@_make_m_preamble_ix:n
2813   { \@@_make_m_preamble:n { #1 } }
2814 }
2815 \cs_new_protected:Npn \@@_make_m_preamble_ix:n #1
2816 {
2817   \tl_gput_right:Nn \g_@@_preamble_tl { < { #1 } }
2818   \@@_make_m_preamble_x:n
2819 }

```

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).

```

2820 \cs_new_protected:Npn \@@_put_box_in_flow:
2821 {
2822   \box_set_ht:Nn \l_tmpa_box { \box_ht:N \l_tmpa_box + \l_tmpa_dim }
2823   \box_set_dp:Nn \l_tmpa_box { \box_dp:N \l_tmpa_box + \l_tmpb_dim }
2824   \tl_if_eq:NnTF \l_@@_baseline_tl { c }
2825   { \box_use_drop:N \l_tmpa_box }
2826   \@@_put_box_in_flow_i:
2827 }

```

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).

```

2828 \cs_new_protected:Npn \@@_put_box_in_flow_i:
2829 {
2830   \pgfpicture
2831   \@@_qpoint:n { row - 1 }
2832   \dim_gset_eq:NN \g_tmpa_dim \pgf@y
2833   \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
2834   \dim_gadd:Nn \g_tmpa_dim \pgf@y
2835   \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).

```

2836     \str_if_in:NnTF \l_@@_baseline_tl { line- }
2837     {
2838         \int_set:Nn \l_tmpa_int
2839         {
2840             \str_range:Nnn
2841             \l_@@_baseline_tl
2842             6
2843             { \tl_count:V \l_@@_baseline_tl }
2844         }
2845         \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
2846     }
2847     {
2848         \str_case:VnF \l_@@_baseline_tl
2849         {
2850             { t } { \int_set:Nn \l_tmpa_int 1 }
2851             { b } { \int_set_eq:NN \l_tmpa_int \c@iRow }
2852         }
2853         { \int_set:Nn \l_tmpa_int \l_@@_baseline_tl }
2854         \bool_lazy_or:nnT
2855         { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
2856         { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
2857         {
2858             \@@_error:n { bad~value~for~baseline }
2859             \int_set:Nn \l_tmpa_int 1
2860         }
2861         \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }

```

We take into account the position of the mathematical axis.

```

2862         \dim_gsub:Nn \g_tmpa_dim { \fontdimen22 \textfont2 }
2863     }
2864     \dim_gsub:Nn \g_tmpa_dim \pgf@y

```

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

```

2865     \endpgfpicture
2866     \box_move_up:nn \g_tmpa_dim { \box_use_drop:N \l_tmpa_box }
2867     \box_use_drop:N \l_tmpa_box
2868 }

```

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).

```

2869 \cs_new_protected:Npn \@@_use_arraybox_with_notes_c:
2870 {

```

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.

```

2871     \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
2872     {
2873         \int_compare:nNnT \c@jCol > 1 % added 2023-08-13
2874         {
2875             \box_set_wd:Nn \l_@@_the_array_box
2876             { \box_wd:N \l_@@_the_array_box - \arraycolsep }
2877         }
2878     }

```

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

```

2879     \begin { minipage } [ t ] { \box_wd:N \l_@@_the_array_box }
2880     \bool_if:NT \l_@@_caption_above_bool
2881     {
2882         \tl_if_empty:NF \l_@@_caption_tl

```

```

2883     {
2884         \bool_set_false:N \g_@@_caption_finished_bool
2885         \int_gzero:N \c@tabularnote
2886         \@@_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).

```

2887         \int_compare:nNnT \g_@@_notes_caption_int > 0
2888         {
2889             \tl_gput_right:Nx \g_@@_aux_tl
2890             {
2891                 \tl_set:Nn \exp_not:N \l_@@_note_in_caption_tl
2892                 { \int_use:N \g_@@_notes_caption_int }
2893             }
2894             \int_gzero:N \g_@@_notes_caption_int
2895         }
2896     }
2897 }

```

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

```

2898     \hbox
2899     {
2900         \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.

```

2901         \@@_create_extra_nodes:
2902         \seq_if_empty:NF \g_@@_blocks_seq \@@_draw_blocks:
2903     }

```

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 it compiles several times its tabular).

```

2904     \bool_lazy_any:nT
2905     {
2906         { ! \seq_if_empty_p:N \g_@@_notes_seq }
2907         { ! \seq_if_empty_p:N \g_@@_notes_in_caption_seq }
2908         { ! \tl_if_empty_p:V \g_@@_tabularnote_tl }
2909     }
2910     \@@_insert_tabularnotes:
2911     \cs_set_eq:NN \tabularnote \@@_tabularnote_error:n
2912     \bool_if:NF \l_@@_caption_above_bool \@@_insert_caption:
2913     \end { minipage }
2914 }

```

```

2915 \cs_new_protected:Npn \@@_insert_caption:
2916 {
2917     \tl_if_empty:NF \l_@@_caption_tl
2918     {
2919         \cs_if_exist:NTF \@capytype
2920         { \@@_insert_caption_i: }
2921         { \@@_error:n { caption-outside-float } }
2922     }
2923 }

```

```

2924 \cs_new_protected:Npn \@@_insert_caption_i:
2925 {
2926     \group_begin:

```

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

```
2927 \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.

```
2928 \IfPackageLoadedTF { floatrow }
2929 { \cs_set_eq:NN \@makecaption \FR@makecaption }
2930 { }
2931 \tl_if_empty:NTF \l_@@_short_caption_tl
2932 { \caption }
2933 { \caption [ \l_@@_short_caption_tl ] }
2934 { \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.

```
2935 \bool_if:NF \g_@@_caption_finished_bool % added 2023/06/30
2936 {
2937   \bool_gset_true:N \g_@@_caption_finished_bool
2938   \int_gset_eq:NN \g_@@_notes_caption_int \c@tabularnote
2939   \int_gzero:N \c@tabularnote
2940 }
2941 \tl_if_empty:NF \l_@@_label_tl { \label { \l_@@_label_tl } }
2942 \group_end:
2943 }
2944 \cs_new_protected:Npn \@_tabularnote_error:n #1
2945 {
2946   \@@_error_or_warning:n { tabularnote~below~the~tabular }
2947   \@@_gredirect_none:n { tabularnote~below~the~tabular }
2948 }
2949 \cs_new_protected:Npn \@_insert_tabularnotes:
2950 {
2951   \seq_gconcat:NNN \g_@@_notes_seq \g_@@_notes_in_caption_seq \g_@@_notes_seq
2952   \int_set:Nn \c@tabularnote { \seq_count:N \g_@@_notes_seq }
2953   \skip_vertical:N 0.65ex
```

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

```
2954 \group_begin:
2955 \l_@@_notes_code_before_tl
2956 \tl_if_empty:NF \g_@@_tabularnote_tl
2957 {
2958   \g_@@_tabularnote_tl \par
2959   \tl_gclear:N \g_@@_tabularnote_tl
2960 }
```

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.

```
2961 \int_compare:nNnT \c@tabularnote > 0
2962 {
2963   \bool_if:NTF \l_@@_notes_para_bool
2964   {
2965     \begin { tabularnotes* }
2966     \seq_map_inline:Nn \g_@@_notes_seq
2967     { \@@_one_tabularnote:nn ##1 }
2968     \strut
2969     \end { tabularnotes* }
```

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

```

2970      \par
2971    }
2972    {
2973      \tabularnotes
2974        \seq_map_inline:Nn \g_@@_notes_seq
2975        { \@@_one_tabularnote:nn ##1 }
2976      \strut
2977      \endtabularnotes
2978    }
2979  }
2980  \unskip
2981  \group_end:
2982  \bool_if:NT \l_@@_notes_bottomrule_bool
2983  {
2984    \IfPackageLoadedTF { booktabs }
2985    {

```

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

```

2986      \skip_vertical:N \aboverulesep
\CT@arc@ is the specification of color defined by colortbl but you use it even if colortbl is not loaded.
2987      { \CT@arc@ \hrule height \heavyrulewidth }
2988    }
2989    { \@@_error_or_warning:n { bottomrule-without-booktabs } }
2990  }
2991  \l_@@_notes_code_after_tl
2992  \seq_gclear:N \g_@@_notes_seq
2993  \seq_gclear:N \g_@@_notes_in_caption_seq
2994  \int_gzero:N \c@tabularnote
2995  }

```

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.

```

2996  \cs_set_protected:Npn \@@_one_tabularnote:nn #1
2997  {
2998    \tl_if_novalue:nTF { #1 }
2999    { \item }
3000    { \item [ \@@_notes_label_in_list:n { #1 } ] }
3001  }

```

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`.

```

3002  \cs_new_protected:Npn \@@_use_arraybox_with_notes_b:
3003  {
3004    \pgfpicture
3005      \@@_qpoint:n { row - 1 }
3006      \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3007      \@@_qpoint:n { row - \int_use:N \c@iRow - base }
3008      \dim_gsub:Nn \g_tmpa_dim \pgf@y
3009    \endpgfpicture
3010    \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3011    \int_compare:nNnT \l_@@_first_row_int = 0
3012    {
3013      \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3014      \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3015    }
3016    \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3017  }

```

Now, the general case.

```
3018 \cs_new_protected:Npn \@@_use_arraybox_with_notes:
3019 {
```

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

```
3020 \tl_if_eq:NnT \l_@@_baseline_tl { t }
3021 { \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`.

```
3022 \pgfpicture
3023 \@@_qpoint:n { row - 1 }
3024 \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3025 \str_if_in:NnTF \l_@@_baseline_tl { line- }
3026 {
3027   \int_set:Nn \l_tmpa_int
3028   {
3029     \str_range:Nnn
3030     \l_@@_baseline_tl
3031     6
3032     { \tl_count:V \l_@@_baseline_tl }
3033   }
3034   \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
3035 }
3036 {
3037   \int_set:Nn \l_tmpa_int \l_@@_baseline_tl
3038   \bool_lazy_or:nnT
3039   { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
3040   { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
3041   {
3042     \@@_error:n { bad-value-for-baseline }
3043     \int_set:Nn \l_tmpa_int 1
3044   }
3045   \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }
3046 }
3047 \dim_gsub:Nn \g_tmpa_dim \pgf@y
3048 \endpgfpicture
3049 \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3050 \int_compare:nNnT \l_@@_first_row_int = 0
3051 {
3052   \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3053   \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3054 }
3055 \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3056 }
```

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.

```
3057 \cs_new_protected:Npn \@@_put_box_in_flow_bis:nn #1 #2
3058 {
```

We will compute the real width of both delimiters used.

```
3059 \dim_zero_new:N \l_@@_real_left_delim_dim
3060 \dim_zero_new:N \l_@@_real_right_delim_dim
3061 \hbox_set:Nn \l_tmpb_box
3062 {
3063   \c_math_toggle_token
3064   \left #1
3065   \vcenter
3066   {
3067     \vbox_to_ht:nn
3068     { \box_ht_plus_dp:N \l_tmpa_box }
3069     { }
```

```

3070     }
3071     \right .
3072     \c_math_toggle_token
3073   }
3074   \dim_set:Nn \l_@@_real_left_delim_dim
3075   { \box_wd:N \l_tmpb_box - \nulldelimiterspace }
3076   \hbox_set:Nn \l_tmpb_box
3077   {
3078     \c_math_toggle_token
3079     \left .
3080     \vbox_to_ht:nn
3081     { \box_ht_plus_dp:N \l_tmpa_box }
3082     { }
3083     \right #2
3084     \c_math_toggle_token
3085   }
3086   \dim_set:Nn \l_@@_real_right_delim_dim
3087   { \box_wd:N \l_tmpb_box - \nulldelimiterspace }

```

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

```

3088   \skip_horizontal:N \l_@@_left_delim_dim
3089   \skip_horizontal:N -\l_@@_real_left_delim_dim
3090   \@@_put_box_in_flow:
3091   \skip_horizontal:N \l_@@_right_delim_dim
3092   \skip_horizontal:N -\l_@@_real_right_delim_dim
3093 }

```

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).

```

3094 \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).

```

3095 {
3096   \peek_remove_spaces:n
3097   {
3098     \peek_meaning:NTF \end
3099     \@@_analyze_end:Nn
3100     {
3101       \@@_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`).

```

3102       \@@_array:V \g_@@_array_preamble_tl
3103     }
3104   }
3105 }
3106 {
3107   \@@_create_col_nodes:
3108   \endarray
3109 }

```

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

```

3110 \NewDocumentEnvironment { @@-light-syntax } { b }
3111 {

```

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.

```

3112 \tl_if_empty:nT { #1 } { @@_fatal:n { empty-environment } }
3113 \tl_map_inline:nn { #1 }
3114 {
3115   \str_if_eq:nnT { ##1 } { & }
3116   { @@_fatal:n { ampersand-in-light-syntax } }
3117   \str_if_eq:nnT { ##1 } { \ }
3118   { @@_fatal:n { double-backslash-in-light-syntax } }
3119 }

```

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`.

```

3120 @@_light_syntax_i:w #1 \CodeAfter \q_stop

```

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

```

3121 }

```

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.

```

3122 {
3123   @@_create_col_nodes:
3124   \endarray
3125 }
3126 \cs_new_protected:Npn @@_light_syntax_i:w #1\CodeAfter #2\q_stop
3127 {
3128   \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).

```

3129 \seq_clear_new:N \l_@@_rows_seq

```

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

```

3130 \tl_set_rescan:Nno \l_@@_end_of_row_tl { } \l_@@_end_of_row_tl
3131 \seq_set_split:Nvn \l_@@_rows_seq \l_@@_end_of_row_tl { #1 }

```

We delete the last row if it is empty.

```

3132 \seq_pop_right:NN \l_@@_rows_seq \l_tmpa_tl
3133 \tl_if_empty:NF \l_tmpa_tl
3134 { \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.

```

3135 \int_compare:nNnT \l_@@_last_row_int = { -1 }
3136 { \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`).

```

3137 \tl_clear_new:N \l_@@_new_body_tl
3138 \int_zero_new:N \l_@@_nb_cols_int

```

First, we treat the first row.

```

3139 \seq_pop_left:NN \l_@@_rows_seq \l_tmpa_tl
3140 @@_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).

```

3141 \seq_map_inline:Nn \l_@@_rows_seq
3142 {
3143   \tl_put_right:Nn \l_@@_new_body_tl { \\ }
3144   \@@_line_with_light_syntax:n { ##1 }
3145 }
3146 \int_compare:nNnT \l_@@_last_col_int = { -1 }
3147 {
3148   \int_set:Nn \l_@@_last_col_int
3149   { \l_@@_nb_cols_int - 1 + \l_@@_first_col_int }
3150 }

```

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.

```

3151 \@@_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`).

```

3152 \@@_array:V \g_@@_array_preamble_tl \l_@@_new_body_tl
3153 }
3154 \cs_new_protected:Npn \@@_line_with_light_syntax:n #1
3155 {
3156   \seq_clear_new:N \l_@@_cells_seq
3157   \seq_set_split:Nnn \l_@@_cells_seq { ~ } { #1 }
3158   \int_set:Nn \l_@@_nb_cols_int
3159   {
3160     \int_max:nn
3161     \l_@@_nb_cols_int
3162     { \seq_count:N \l_@@_cells_seq }
3163   }
3164   \seq_pop_left:NN \l_@@_cells_seq \l_tmpa_tl
3165   \tl_put_right:NV \l_@@_new_body_tl \l_tmpa_tl
3166   \seq_map_inline:Nn \l_@@_cells_seq
3167   { \tl_put_right:Nn \l_@@_new_body_tl { & ##1 } }
3168 }
3169 \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`.

```

3170 \cs_new_protected:Npn \@@_analyze_end:Nn #1 #2
3171 {
3172   \str_if_eq:VnT \g_@@_name_env_str { #2 }
3173   { \@@_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.

```

3174 \end { #2 }
3175 }

```

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).

```

3176 \cs_new:Npn \@@_create_col_nodes:
3177 {
3178   \crrc
3179   \int_compare:nNnT \l_@@_first_col_int = 0
3180   {
3181     \omit
3182     \hbox_overlap_left:n
3183     {

```

```

3184         \bool_if:NT \l_@@_code_before_bool
3185         { \pgfsys@markposition { \@@_env: - col - 0 } }
3186     \pgfpicture
3187     \pgfrememberpicturerepositiononpagetrue
3188     \pgfcoordinate { \@@_env: - col - 0 } \pgfpointorigin
3189     \str_if_empty:NF \l_@@_name_str
3190     { \pgfnodealias { \l_@@_name_str - col - 0 } { \@@_env: - col - 0 } }
3191     \endpgfpicture
3192     \skip_horizontal:N 2\col@sep
3193     \skip_horizontal:N \g_@@_width_first_col_dim
3194 }
3195 &
3196 }
3197 \omit

```

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

```

3198     \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`).

```

3199     \int_compare:nNnTF \l_@@_first_col_int = 0
3200     {
3201         \bool_if:NT \l_@@_code_before_bool
3202         {
3203             \hbox
3204             {
3205                 \skip_horizontal:N -0.5\arrayrulewidth
3206                 \pgfsys@markposition { \@@_env: - col - 1 }
3207                 \skip_horizontal:N 0.5\arrayrulewidth
3208             }
3209         }
3210     \pgfpicture
3211     \pgfrememberpicturerepositiononpagetrue
3212     \pgfcoordinate { \@@_env: - col - 1 }
3213     { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3214     \str_if_empty:NF \l_@@_name_str
3215     { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
3216     \endpgfpicture
3217 }
3218 {
3219     \bool_if:NT \l_@@_code_before_bool
3220     {
3221         \hbox
3222         {
3223             \skip_horizontal:N 0.5\arrayrulewidth
3224             \pgfsys@markposition { \@@_env: - col - 1 }
3225             \skip_horizontal:N -0.5\arrayrulewidth
3226         }
3227     }
3228     \pgfpicture
3229     \pgfrememberpicturerepositiononpagetrue
3230     \pgfcoordinate { \@@_env: - col - 1 }
3231     { \pgfpoint { 0.5 \arrayrulewidth } \c_zero_dim }
3232     \str_if_empty:NF \l_@@_name_str
3233     { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
3234     \endpgfpicture
3235 }

```

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.

```

3236 \skip_gset:Nn \g_tmpa_skip { 0 pt~plus 1 fill }
3237 \bool_if:NF \l_@@_auto_columns_width_bool
3238 { \dim_compare:nNnT \l_@@_columns_width_dim > \c_zero_dim }
3239 {
3240   \bool_lazy_and:nnTF
3241     \l_@@_auto_columns_width_bool
3242     { \bool_not_p:n \l_@@_block_auto_columns_width_bool }
3243     { \skip_gset_eq:NN \g_tmpa_skip \g_@@_max_cell_width_dim }
3244     { \skip_gset_eq:NN \g_tmpa_skip \l_@@_columns_width_dim }
3245     \skip_gadd:Nn \g_tmpa_skip { 2 \col@sep }
3246 }
3247 \skip_horizontal:N \g_tmpa_skip
3248 \hbox
3249 {
3250   \bool_if:NT \l_@@_code_before_bool
3251   {
3252     \hbox
3253     {
3254       \skip_horizontal:N -0.5\arrayrulewidth
3255       \pgfsys@markposition { \@@_env: - col - 2 }
3256       \skip_horizontal:N 0.5\arrayrulewidth
3257     }
3258   }
3259   \pgfpicture
3260   \pgfrememberpicturepositiononpagetrue
3261   \pgfcoordinate { \@@_env: - col - 2 }
3262   { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3263   \str_if_empty:NF \l_@@_name_str
3264   { \pgfnodealias { \l_@@_name_str - col - 2 } { \@@_env: - col - 2 } }
3265   \endpgfpicture
3266 }

```

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.

```

3267 \int_gset:Nn \g_tmpa_int 1
3268 \bool_if:NTF \g_@@_last_col_found_bool
3269 { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 3 } 0 } }
3270 { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 2 } 0 } }
3271 {
3272   &
3273   \omit
3274   \int_gincr:N \g_tmpa_int

```

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

```

3275 \skip_horizontal:N \g_tmpa_skip
3276 \bool_if:NT \l_@@_code_before_bool
3277 {
3278   \hbox
3279   {
3280     \skip_horizontal:N -0.5\arrayrulewidth
3281     \pgfsys@markposition
3282     { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3283     \skip_horizontal:N 0.5\arrayrulewidth
3284   }
3285 }

```

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

```

3286 \pgfpicture
3287 \pgfrememberpicturepositiononpagetrue
3288 \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3289 { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3290 \str_if_empty:NF \l_@@_name_str
3291 {
3292   \pgfnodealias
3293   { \l_@@_name_str - col - \int_eval:n { \g_tmpa_int + 1 } }

```

```

3294         { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3295     }
3296 \endpgfpicture
3297 }

3298 &
3299 \omit

```

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

```

3300 \int_compare:nNt \g_@@_col_total_int = 1
3301 { \skip_gset:Nn \g_tmpa_skip { 0 pt~plus 1 fill } }
3302 \skip_horizontal:N \g_tmpa_skip
3303 \int_gincr:N \g_tmpa_int
3304 \bool_lazy_all:nT
3305 {
3306     { \bool_not_p:n \g_@@_delims_bool }
3307     { \bool_not_p:n \l_@@_tabular_bool }
3308     { \clist_if_empty_p:N \l_@@_vlines_clist }
3309     { \bool_not_p:n \l_@@_exterior_arraycolsep_bool }
3310     { ! \l_@@_bar_at_end_of_pream_bool }
3311 }
3312 { \skip_horizontal:N -\col@sep }
3313 \bool_if:NT \l_@@_code_before_bool
3314 {
3315     \hbox
3316     {
3317         \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.

```

3318         \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3319         { \skip_horizontal:N -\arraycolsep }
3320 \pgfsys@markposition
3321 { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3322 \skip_horizontal:N 0.5\arrayrulewidth
3323 \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3324 { \skip_horizontal:N \arraycolsep }
3325 }
3326 }
3327 \pgfpicture
3328 \pgfrememberpicturepositiononpagetrue
3329 \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3330 {
3331     \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3332     {
3333         \pgfpoint
3334         { - 0.5 \arrayrulewidth - \arraycolsep }
3335         \c_zero_dim
3336     }
3337     { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3338 }
3339 \str_if_empty:NF \l_@@_name_str
3340 {
3341     \pgfnodealias
3342     { \l_@@_name_str - col - \int_eval:n { \g_tmpa_int + 1 } }
3343     { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3344 }
3345 \endpgfpicture

3346 \bool_if:NT \g_@@_last_col_found_bool
3347 {

```

```

3348 \hbox_overlap_right:n
3349 {
3350   \skip_horizontal:N \g_@@_width_last_col_dim
3351   \bool_if:NT \l_@@_code_before_bool
3352   {
3353     \pgfsys@markposition
3354     { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3355   }
3356   \pgfpicture
3357   \pgfrememberpicturepositiononpagetrue
3358   \pgfcoordinate
3359   { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3360   \pgfpointorigin
3361   \str_if_empty:NF \l_@@_name_str
3362   {
3363     \pgfnodealias
3364     {
3365       \l_@@_name_str - col
3366       - \int_eval:n { \g_@@_col_total_int + 1 }
3367     }
3368     { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3369   }
3370   \endpgfpicture
3371 }
3372 }
3373 \cr
3374 }

```

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

```

3375 \tl_const:Nn \c_@@_preamble_first_col_tl
3376 {
3377   >
3378   {

```

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).

```

3379   \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
3380   \bool_gset_true:N \g_@@_after_col_zero_bool
3381   \@@_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.

```

3382   \hbox_set:Nw \l_@@_cell_box
3383   \@@_math_toggle_token:
3384   \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”.

```

3385   \bool_lazy_and:nnT
3386   { \int_compare_p:nNn \c@iRow > 0 }
3387   {
3388     \bool_lazy_or_p:nn
3389     { \int_compare_p:nNn \l_@@_last_row_int < 0 }
3390     { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
3391   }
3392   {
3393     \l_@@_code_for_first_col_tl
3394     \xglobal \colorlet { nicematrix-first-col } { . }
3395   }
3396 }

```

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`.

```

3397   1
3398   <
3399   {
3400     \@@_math_toggle_token:
3401     \hbox_set_end:
3402     \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
3403     \@@_adjust_size_box:
3404     \@@_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.

```

3405     \dim_gset:Nn \g_@@_width_first_col_dim
3406     { \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.

```

3407     \hbox_overlap_left:n
3408     {
3409       \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > \c_zero_dim
3410       \@@_node_for_cell:
3411       { \box_use_drop:N \l_@@_cell_box }
3412       \skip_horizontal:N \l_@@_left_delim_dim
3413       \skip_horizontal:N \l_@@_left_margin_dim
3414       \skip_horizontal:N \l_@@_extra_left_margin_dim
3415     }
3416     \bool_gset_false:N \g_@@_empty_cell_bool
3417     \skip_horizontal:N -2\col@sep
3418   }
3419 }

```

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

```

3420 \tl_const:Nn \c_@@_preamble_last_col_tl
3421 {
3422   >
3423   {
3424     \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).

```

3425     \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.

```

3426     \bool_gset_true:N \g_@@_last_col_found_bool
3427     \int_gincr:N \c@jCol
3428     \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.

```

3429     \hbox_set:Nw \l_@@_cell_box
3430     \@@_math_toggle_token:
3431     \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”.

```

3432     \int_compare:nNnT \c@iRow > 0
3433     {
3434       \bool_lazy_or:nnT
3435       { \int_compare_p:nNn \l_@@_last_row_int < 0 }
3436       { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
3437       {
3438         \l_@@_code_for_last_col_tl
3439         \xglobal \colorlet { nicematrix-last-col } { . }
3440       }
3441     }
3442   }
3443   1
3444   <

```

```

3445 {
3446   \@@_math_toggle_token:
3447   \hbox_set_end:
3448   \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
3449   \@@_adjust_size_box:
3450   \@@_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.

```

3451   \dim_gset:Nn \g_@@_width_last_col_dim
3452   { \dim_max:nn \g_@@_width_last_col_dim { \box_wd:N \l_@@_cell_box } }
3453   \skip_horizontal:N -2\col@sep

```

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

```

3454   \hbox_overlap_right:n
3455   {
3456     \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \c_zero_dim
3457     {
3458       \skip_horizontal:N \l_@@_right_delim_dim
3459       \skip_horizontal:N \l_@@_right_margin_dim
3460       \skip_horizontal:N \l_@@_extra_right_margin_dim
3461       \@@_node_for_cell:
3462     }
3463   }
3464   \bool_gset_false:N \g_@@_empty_cell_bool
3465 }
3466 }

```

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

```

3467 \NewDocumentEnvironment { NiceArray } { }
3468 {
3469   \bool_gset_false:N \g_@@_delims_bool
3470   \str_if_empty:NT \g_@@_name_env_str
3471   { \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).

```

3472   \NiceArrayWithDelims . .
3473 }
3474 { \endNiceArrayWithDelims }

```

We create the variants of the environment {NiceArrayWithDelims}.

```

3475 \cs_new_protected:Npn \@@_def_env:nnn #1 #2 #3
3476 {
3477   \NewDocumentEnvironment { #1 NiceArray } { }
3478   {
3479     \bool_gset_true:N \g_@@_delims_bool
3480     \str_if_empty:NT \g_@@_name_env_str
3481     { \str_gset:Nn \g_@@_name_env_str { #1 NiceArray } }
3482     \@@_test_if_math_mode:
3483     \NiceArrayWithDelims #2 #3
3484   }
3485   { \endNiceArrayWithDelims }
3486 }
3487 \@@_def_env:nnn p ( )
3488 \@@_def_env:nnn b [ ]
3489 \@@_def_env:nnn B \{ \}
3490 \@@_def_env:nnn v | |
3491 \@@_def_env:nnn V \| \|

```

14 The environment {NiceMatrix} and its variants

```

3492 \cs_new_protected:Npn \@@_begin_of_NiceMatrix:nn #1 #2
3493 {
3494   \bool_set_false:N \l_@@_preamble_bool
3495   \tl_clear:N \l_tmpa_tl
3496   \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3497     { \tl_set:Nn \l_tmpa_tl { @ { } } }
3498   \tl_put_right:Nn \l_tmpa_tl
3499     {
3500       *
3501       {
3502         \int_case:nnF \l_@@_last_col_int
3503         {
3504           { -2 } { \c@MaxMatrixCols }
3505           { -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).

```

3506       }
3507       { \int_eval:n { \l_@@_last_col_int - 1 } }
3508     }
3509     { #2 }
3510   }
3511   \tl_set:Nn \l_tmpb_tl { \use:c { #1 NiceArray } }
3512   \exp_args:NV \l_tmpb_tl \l_tmpa_tl
3513 }
3514 \cs_generate_variant:Nn \@@_begin_of_NiceMatrix:nn { n V }
3515 \clist_map_inline:nn { p , b , B , v , V }
3516 {
3517   \NewDocumentEnvironment { #1 NiceMatrix } { ! 0 { } }
3518   {
3519     \bool_gset_true:N \g_@@_delims_bool
3520     \str_gset:Nn \g_@@_name_env_str { #1 NiceMatrix }
3521     \keys_set:nn { NiceMatrix / NiceMatrix } { ##1 }
3522     \@@_begin_of_NiceMatrix:nV { #1 } \l_@@_columns_type_tl
3523   }
3524   { \use:c { end #1 NiceArray } }
3525 }

```

We define also an environment {NiceMatrix}

```

3526 \NewDocumentEnvironment { NiceMatrix } { ! 0 { } }
3527 {
3528   \str_gset:Nn \g_@@_name_env_str { NiceMatrix }
3529   \keys_set:nn { NiceMatrix / NiceMatrix } { #1 }
3530   \bool_lazy_or:nnT
3531     { \clist_if_empty_p:N \l_@@_vlines_clist }
3532     { \l_@@_except_borders_bool }
3533     { \bool_set_true:N \l_@@_NiceMatrix_without_vlines_bool }
3534   \@@_begin_of_NiceMatrix:nV { } \l_@@_columns_type_tl
3535 }
3536 { \endNiceArray }

```

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

```

3537 \cs_new_protected:Npn \@@_NotEmpty:
3538 { \bool_gset_true:N \g_@@_not_empty_cell_bool }

```

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

```

3539 \NewDocumentEnvironment { NiceTabular } { 0 { } m ! 0 { } }
3540 {

```


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`.

```

3541 \dim_compare:nNnT \l_@@_width_dim = \c_zero_dim
3542 { \dim_set_eq:NN \l_@@_width_dim \linewidth }
3543 \str_gset:Nn \g_@@_name_env_str { NiceTabular }
3544 \keys_set:nn { NiceMatrix / NiceTabular } { #1 , #3 }
3545 \tl_if_empty:NF \l_@@_short_caption_tl
3546 {
3547   \tl_if_empty:NT \l_@@_caption_tl
3548   {
3549     \@@_error_or_warning:n { short-caption~without~caption }
3550     \tl_set_eq:NN \l_@@_caption_tl \l_@@_short_caption_tl
3551   }
3552 }
3553 \tl_if_empty:NF \l_@@_label_tl
3554 {
3555   \tl_if_empty:NT \l_@@_caption_tl
3556   { \@@_error_or_warning:n { label~without~caption } }
3557 }
3558 \NewDocumentEnvironment { TabularNote } { b }
3559 {
3560   \bool_if:NTF \l_@@_in_code_after_bool
3561   { \@@_error_or_warning:n { TabularNote~in~CodeAfter } }
3562   {
3563     \tl_if_empty:NF \g_@@_tabularnote_tl
3564     { \tl_gput_right:Nn \g_@@_tabularnote_tl { \par } }
3565     \tl_gput_right:Nn \g_@@_tabularnote_tl { ##1 }
3566   }
3567 }
3568 { }
3569 \bool_set_true:N \l_@@_tabular_bool
3570 \NiceArray { #2 }
3571 }
3572 { \endNiceArray }

3573 \NewDocumentEnvironment { NiceTabularX } { m O { } m ! O { } }
3574 {
3575   \str_gset:Nn \g_@@_name_env_str { NiceTabularX }
3576   \dim_zero_new:N \l_@@_width_dim
3577   \dim_set:Nn \l_@@_width_dim { #1 }
3578   \keys_set:nn { NiceMatrix / NiceTabular } { #2 , #4 }
3579   \bool_set_true:N \l_@@_tabular_bool
3580   \NiceArray { #3 }
3581 }
3582 {
3583   \endNiceArray
3584   \int_compare:nNnT \g_@@_total_X_weight_int = \c_zero_int
3585   { \@@_error:n { NiceTabularX~without~X } }
3586 }

3587 \NewDocumentEnvironment { NiceTabular* } { m O { } m ! O { } }
3588 {
3589   \str_gset:Nn \g_@@_name_env_str { NiceTabular* }
3590   \dim_set:Nn \l_@@_tabular_width_dim { #1 }
3591   \keys_set:nn { NiceMatrix / NiceTabular } { #2 , #4 }
3592   \bool_set_true:N \l_@@_tabular_bool
3593   \NiceArray { #3 }
3594 }
3595 { \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`).

```

3596 \cs_new_protected:Npn \@@_deal_with_rounded_corners:
3597 {
3598   \bool_lazy_all:nT
3599   {
3600     { \int_compare_p:nNn \l_@@_tab_rounded_corners_dim > \c_zero_dim }
3601     \l_@@_hvlines_bool
3602     { ! \g_@@_delims_bool }
3603     { ! \l_@@_except_borders_bool }
3604   }
3605   {
3606     \bool_set_true:N \l_@@_except_borders_bool
3607     \clist_if_empty:NF \l_@@_corners_clist
3608     { \@@_error:n { hvlines,~rounded-corners~and~corners } }
3609     \tl_gput_right:Nn \g_@@_pre_code_after_tl
3610     {
3611       \@@_stroke_block:nnn
3612       {
3613         rounded-corners = \dim_use:N \l_@@_tab_rounded_corners_dim ,
3614         draw = \l_@@_rules_color_tl
3615       }
3616       { 1-1 }
3617       { \int_use:N \c@iRow - \int_use:N \c@jCol }
3618     }
3619   }
3620 }

3621 \cs_new_protected:Npn \@@_after_array:
3622 {
3623   \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.

```

3624   \bool_if:NT \g_@@_last_col_found_bool
3625   { \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`.

```

3626   \bool_if:NT \l_@@_last_col_without_value_bool
3627   { \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.

```

3628   \bool_if:NT \l_@@_last_row_without_value_bool
3629   { \int_set_eq:NN \l_@@_last_row_int \g_@@_row_total_int }

3630   \tl_gput_right:Nx \g_@@_aux_tl
3631   {
3632     \seq_gset_from_clist:Nn \exp_not:N \g_@@_size_seq
3633     {
3634       \int_use:N \l_@@_first_row_int ,
3635       \int_use:N \c@iRow ,
3636       \int_use:N \g_@@_row_total_int ,

```

```

3637         \int_use:N \l_@@_first_col_int ,
3638         \int_use:N \c@jCol ,
3639         \int_use:N \g_@@_col_total_int
3640     }
3641 }

```

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`).

```

3642 \seq_if_empty:NF \g_@@_pos_of_blocks_seq
3643 {
3644     \tl_gput_right:Nx \g_@@_aux_tl
3645     {
3646         \seq_gset_from_clist:Nn \exp_not:N \g_@@_pos_of_blocks_seq
3647         { \seq_use:Nnnn \g_@@_pos_of_blocks_seq , , , }
3648     }
3649 }
3650 \seq_if_empty:NF \g_@@_multicolumn_cells_seq
3651 {
3652     \tl_gput_right:Nx \g_@@_aux_tl
3653     {
3654         \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_cells_seq
3655         { \seq_use:Nnnn \g_@@_multicolumn_cells_seq , , , }
3656         \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_sizes_seq
3657         { \seq_use:Nnnn \g_@@_multicolumn_sizes_seq , , , }
3658     }
3659 }

```

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

```

3660 \@@_create_diag_nodes:

```

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

```

3661 \pgfpicture
3662 \int_step_inline:nn \c@iRow
3663 {
3664     \pgfnodealias
3665     { \@@_env: - ##1 - last }
3666     { \@@_env: - ##1 - \int_use:N \c@jCol }
3667 }
3668 \int_step_inline:nn \c@jCol
3669 {
3670     \pgfnodealias
3671     { \@@_env: - last - ##1 }
3672     { \@@_env: - \int_use:N \c@iRow - ##1 }
3673 }
3674 \str_if_empty:NF \l_@@_name_str
3675 {
3676     \int_step_inline:nn \c@iRow
3677     {
3678         \pgfnodealias
3679         { \l_@@_name_str - ##1 - last }
3680         { \@@_env: - ##1 - \int_use:N \c@jCol }
3681     }
3682     \int_step_inline:nn \c@jCol
3683     {
3684         \pgfnodealias
3685         { \l_@@_name_str - last - ##1 }
3686         { \@@_env: - \int_use:N \c@iRow - ##1 }
3687     }
3688 }
3689 \endpgfpicture

```

By default, the diagonal lines will be parallelized¹⁰. 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.

```

3690   \bool_if:NT \l_@@_parallelize_diags_bool
3691   {
3692     \int_gzero_new:N \g_@@_ddots_int
3693     \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.

```

3694     \dim_gzero_new:N \g_@@_delta_x_one_dim
3695     \dim_gzero_new:N \g_@@_delta_y_one_dim
3696     \dim_gzero_new:N \g_@@_delta_x_two_dim
3697     \dim_gzero_new:N \g_@@_delta_y_two_dim
3698   }

3699   \int_zero_new:N \l_@@_initial_i_int
3700   \int_zero_new:N \l_@@_initial_j_int
3701   \int_zero_new:N \l_@@_final_i_int
3702   \int_zero_new:N \l_@@_final_j_int
3703   \bool_set_false:N \l_@@_initial_open_bool
3704   \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.

```

3705   \bool_if:NT \l_@@_small_bool
3706   {
3707     \dim_set:Nn \l_@@_xdots_radius_dim { 0.7 \l_@@_xdots_radius_dim }
3708     \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.

```

3709     \dim_set:Nn \l_@@_xdots_shorten_start_dim
3710     { 0.6 \l_@@_xdots_shorten_start_dim }
3711     \dim_set:Nn \l_@@_xdots_shorten_end_dim
3712     { 0.6 \l_@@_xdots_shorten_end_dim }
3713   }

```

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

```

3714   \@@_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.

```

3715   \@@_compute_corners:

```

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

```

3716   \@@_adjust_pos_of_blocks_seq:
3717   \@@_deal_with_rounded_corners:
3718   \tl_if_empty:NF \l_@@_hlines_clist \@@_draw_hlines:
3719   \tl_if_empty:NF \l_@@_vlines_clist \@@_draw_vlines:

```

¹⁰It’s possible to use the option `parallelize-diags` to disable this parallelization.

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

```

3720 \IfPackageLoadedTF { tikz }
3721 {
3722   \tikzset
3723   {
3724     every~picture / .style =
3725     {
3726       overlay ,
3727       remember~picture ,
3728       name-prefix = \@@_env: -
3729     }
3730   }
3731 }
3732 { }
3733 \cs_set_eq:NN \ialign \@@_old_ialign:
3734 \cs_set_eq:NN \SubMatrix \@@_SubMatrix
3735 \cs_set_eq:NN \UnderBrace \@@_UnderBrace
3736 \cs_set_eq:NN \OverBrace \@@_OverBrace
3737 \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
3738 \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
3739 \cs_set_eq:NN \line \@@_line
3740 \g_@@_pre_code_after_tl
3741 \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.

```

3742 \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).

```

3743 \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`).

```

3744 \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
3745 { \@@_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:`.

```

3746 \bool_set_true:N \l_@@_in_code_after_bool
3747 \exp_last_unbraced:NV \@@_CodeAfter_keys: \g_nicematrix_code_after_tl
3748 \scan_stop:
3749 \tl_gclear:N \g_nicematrix_code_after_tl
3750 \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.

```

3751 \tl_if_empty:NF \g_@@_pre_code_before_tl
3752 {
3753   \tl_gput_right:Nx \g_@@_aux_tl
3754   {
3755     \tl_gset:Nn \exp_not:N \g_@@_pre_code_before_tl
3756     { \exp_not:V \g_@@_pre_code_before_tl }
3757   }
3758   \tl_gclear:N \g_@@_pre_code_before_tl
3759 }
3760 \tl_if_empty:NF \g_nicematrix_code_before_tl
3761 {
3762   \tl_gput_right:Nx \g_@@_aux_tl

```

```

3763     {
3764         \tl_gset:Nn \exp_not:N \g_@@_code_before_tl
3765         { \exp_not:V \g_nicematrix_code_before_tl }
3766     }
3767     \tl_gclear:N \g_nicematrix_code_before_tl
3768 }

```

```

3769 \str_gclear:N \g_@@_name_env_str
3770 \@@_restore_iRow_jCol:

```

The command `\CT@arc@` contains the instruction of color for the rules of the array¹¹. 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`.

```

3771 \cs_gset_eq:NN \CT@arc@ \@@_old_CT@arc@
3772 }

```

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`.

```

3773 \NewDocumentCommand \@@_CodeAfter_keys: { 0 { } }
3774 { \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).

```

3775 \cs_new_protected:Npn \@@_adjust_pos_of_blocks_seq:
3776 {
3777     \seq_gset_map_x:NNn \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq
3778     { \@@_adjust_pos_of_blocks_seq_i:nnnnn #1 }
3779 }

```

The following command must *not* be protected.

```

3780 \cs_new:Npn \@@_adjust_pos_of_blocks_seq_i:nnnnn #1 #2 #3 #4 #5
3781 {
3782     { #1 }
3783     { #2 }
3784     {
3785         \int_compare:nNnTF { #3 } > { 99 }
3786         { \int_use:N \c@iRow }
3787         { #3 }
3788     }
3789     {
3790         \int_compare:nNnTF { #4 } > { 99 }
3791         { \int_use:N \c@jCol }
3792         { #4 }
3793     }
3794     { #5 }
3795 }

```

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).

```

3796 \hook_gput_code:nnn { begindocument } { . }

```

¹¹e.g. `\color[rgb]{0.5,0.5,0}`

```

3797 {
3798   \cs_new_protected:Npn \@@_draw_dotted_lines:
3799   {
3800     \c_@@_pgfortikzpicture_tl
3801     \@@_draw_dotted_lines_i:
3802     \c_@@_endpgfortikzpicture_tl
3803   }
3804 }

```

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

```

3805 \cs_new_protected:Npn \@@_draw_dotted_lines_i:
3806 {
3807   \pgfrememberpicturepositiononpagetrue
3808   \pgf@relevantforpicturesizefalse
3809   \g_@@_HVdotsfor_lines_tl
3810   \g_@@_Vdots_lines_tl
3811   \g_@@_Ddots_lines_tl
3812   \g_@@_Iddots_lines_tl
3813   \g_@@_Cdots_lines_tl
3814   \g_@@_Ldots_lines_tl
3815 }

3816 \cs_new_protected:Npn \@@_restore_iRow_jCol:
3817 {
3818   \cs_if_exist:NT \theiRow { \int_gset_eq:NN \c@iRow \l_@@_old_iRow_int }
3819   \cs_if_exist:NT \thejCol { \int_gset_eq:NN \c@jCol \l_@@_old_jCol_int }
3820 }

```

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

```

3821 \pgfdeclareshape { @@_diag_node }
3822 {
3823   \savedanchor { \five }
3824   {
3825     \dim_gset_eq:NN \pgf@x \l_tmpa_dim
3826     \dim_gset_eq:NN \pgf@y \l_tmpb_dim
3827   }
3828   \anchor { 5 } { \five }
3829   \anchor { center } { \pgfpointorigin }
3830 }

```

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).

```

3831 \cs_new_protected:Npn \@@_create_diag_nodes:
3832 {
3833   \pgfpicture
3834   \pgfrememberpicturepositiononpagetrue
3835   \int_step_inline:nn { \int_max:nn \c@iRow \c@jCol }
3836   {
3837     \@@_qpoint:n { col - \int_min:nn { ##1 } { \c@jCol + 1 } }
3838     \dim_set_eq:NN \l_tmpa_dim \pgf@x
3839     \@@_qpoint:n { row - \int_min:nn { ##1 } { \c@iRow + 1 } }
3840     \dim_set_eq:NN \l_tmpb_dim \pgf@y
3841     \@@_qpoint:n { col - \int_min:nn { ##1 + 1 } { \c@jCol + 1 } }
3842     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
3843     \@@_qpoint:n { row - \int_min:nn { ##1 + 1 } { \c@iRow + 1 } }
3844     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
3845     \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 `@â_diag_node`) that we will construct.

```

3846     \dim_set:Nn \l_tmpa_dim { ( \l_@@_tmpc_dim - \l_tmpa_dim ) / 2 }
3847     \dim_set:Nn \l_tmpb_dim { ( \l_@@_tmpd_dim - \l_tmpb_dim ) / 2 }
3848     \pgfnode { @@_diag_node } { center } { } { \@@_env: - ##1 } { }
3849     \str_if_empty:NF \l_@@_name_str
3850     { \pgfnodealias { \l_@@_name_str - ##1 } { \@@_env: - ##1 } }
3851 }

```

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

```

3852     \int_set:Nn \l_tmpa_int { \int_max:nn \c@iRow \c@jCol + 1 }
3853     \@@_qpoint:n { row - \int_min:nn { \l_tmpa_int } { \c@iRow + 1 } }
3854     \dim_set_eq:NN \l_tmpa_dim \pgf@y
3855     \@@_qpoint:n { col - \int_min:nn { \l_tmpa_int } { \c@jCol + 1 } }
3856     \pgfcoordinate
3857     { \@@_env: - \int_use:N \l_tmpa_int } { \pgfpoint \pgf@x \l_tmpa_dim }
3858     \pgfnodealias
3859     { \@@_env: - last }
3860     { \@@_env: - \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
3861     \str_if_empty:NF \l_@@_name_str
3862     {
3863         \pgfnodealias
3864         { \l_@@_name_str - \int_use:N \l_tmpa_int }
3865         { \@@_env: - \int_use:N \l_tmpa_int }
3866         \pgfnodealias
3867         { \l_@@_name_str - last }
3868         { \@@_env: - last }
3869     }
3870     \endpgfpicture
3871 }

```

17 We draw the dotted lines

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

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

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

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

This command computes:

- `\l_@@_initial_i_int` and `\l_@@_initial_j_int` which are the coordinates of one extremity of the line;
- `\l_@@_final_i_int` and `\l_@@_final_j_int` which are the coordinates of the other extremity of the line;

- \l_@@_initial_open_bool and \l_@@_final_open_bool to indicate whether the extremities are open or not.

```
3872 \cs_new_protected:Npn \@@_find_extremities_of_line:nnnn #1 #2 #3 #4
3873 {
```

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

```
3874 \cs_set:cpn { @@ _ dotted _ #1 - #2 } { }
```

Initialization of variables.

```
3875 \int_set:Nn \l_@@_initial_i_int { #1 }
3876 \int_set:Nn \l_@@_initial_j_int { #2 }
3877 \int_set:Nn \l_@@_final_i_int { #1 }
3878 \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.

```
3879 \bool_set_false:N \l_@@_stop_loop_bool
3880 \bool_do_until:Nn \l_@@_stop_loop_bool
3881 {
3882   \int_add:Nn \l_@@_final_i_int { #3 }
3883   \int_add:Nn \l_@@_final_j_int { #4 }
```

We test if we are still in the matrix.

```
3884 \bool_set_false:N \l_@@_final_open_bool
3885 \int_compare:nNnTF \l_@@_final_i_int > \l_@@_row_max_int
3886 {
3887   \int_compare:nNnTF { #3 } = 1
3888   { \bool_set_true:N \l_@@_final_open_bool }
3889   {
3890     \int_compare:nNnTF \l_@@_final_j_int > \l_@@_col_max_int
3891     { \bool_set_true:N \l_@@_final_open_bool }
3892   }
3893 }
3894 {
3895   \int_compare:nNnTF \l_@@_final_j_int < \l_@@_col_min_int
3896   {
3897     \int_compare:nNnTF { #4 } = { -1 }
3898     { \bool_set_true:N \l_@@_final_open_bool }
3899   }
3900   {
3901     \int_compare:nNnTF \l_@@_final_j_int > \l_@@_col_max_int
3902     {
3903       \int_compare:nNnTF { #4 } = 1
3904       { \bool_set_true:N \l_@@_final_open_bool }
3905     }
3906   }
3907 }
3908 \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.

```
3909 {
```

We do a step backwards.

```
3910 \int_sub:Nn \l_@@_final_i_int { #3 }
3911 \int_sub:Nn \l_@@_final_j_int { #4 }
3912 \bool_set_true:N \l_@@_stop_loop_bool
3913 }
```

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.

```
3914 {
3915   \cs_if_exist:cTF
3916   {
3917     @@ _ dotted _
```

```

3918         \int_use:N \l_@@_final_i_int -
3919         \int_use:N \l_@@_final_j_int
3920     }
3921     {
3922         \int_sub:Nn \l_@@_final_i_int { #3 }
3923         \int_sub:Nn \l_@@_final_j_int { #4 }
3924         \bool_set_true:N \l_@@_final_open_bool
3925         \bool_set_true:N \l_@@_stop_loop_bool
3926     }
3927     {
3928         \cs_if_exist:cTF
3929         {
3930             pgf @ sh @ ns @ \@@_env:
3931             - \int_use:N \l_@@_final_i_int
3932             - \int_use:N \l_@@_final_j_int
3933         }
3934         { \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.

```

3935         {
3936             \cs_set:cpn
3937             {
3938                 @@ _ dotted _
3939                 \int_use:N \l_@@_final_i_int -
3940                 \int_use:N \l_@@_final_j_int
3941             }
3942             { }
3943         }
3944     }
3945 }
3946 }

```

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

```

3947     \bool_set_false:N \l_@@_stop_loop_bool
3948     \bool_do_until:Nn \l_@@_stop_loop_bool
3949     {
3950         \int_sub:Nn \l_@@_initial_i_int { #3 }
3951         \int_sub:Nn \l_@@_initial_j_int { #4 }
3952         \bool_set_false:N \l_@@_initial_open_bool
3953         \int_compare:nNnTF \l_@@_initial_i_int < \l_@@_row_min_int
3954         {
3955             \int_compare:nNnTF { #3 } = 1
3956             { \bool_set_true:N \l_@@_initial_open_bool }
3957             {
3958                 \int_compare:nNnT \l_@@_initial_j_int = { \l_@@_col_min_int -1 }
3959                 { \bool_set_true:N \l_@@_initial_open_bool }
3960             }
3961         }
3962     {
3963         \int_compare:nNnTF \l_@@_initial_j_int < \l_@@_col_min_int
3964         {
3965             \int_compare:nNnT { #4 } = 1
3966             { \bool_set_true:N \l_@@_initial_open_bool }
3967         }
3968     {
3969         \int_compare:nNnT \l_@@_initial_j_int > \l_@@_col_max_int
3970         {

```

```

3971         \int_compare:nNnT { #4 } = { -1 }
3972         { \bool_set_true:N \l_@@_initial_open_bool }
3973     }
3974 }
3975 }
3976 \bool_if:NTF \l_@@_initial_open_bool
3977 {
3978     \int_add:Nn \l_@@_initial_i_int { #3 }
3979     \int_add:Nn \l_@@_initial_j_int { #4 }
3980     \bool_set_true:N \l_@@_stop_loop_bool
3981 }
3982 {
3983     \cs_if_exist:cTF
3984     {
3985         @@ _ dotted _
3986         \int_use:N \l_@@_initial_i_int -
3987         \int_use:N \l_@@_initial_j_int
3988     }
3989     {
3990         \int_add:Nn \l_@@_initial_i_int { #3 }
3991         \int_add:Nn \l_@@_initial_j_int { #4 }
3992         \bool_set_true:N \l_@@_initial_open_bool
3993         \bool_set_true:N \l_@@_stop_loop_bool
3994     }
3995     {
3996         \cs_if_exist:cTF
3997         {
3998             pgf @ sh @ ns @ \@@_env:
3999             - \int_use:N \l_@@_initial_i_int
4000             - \int_use:N \l_@@_initial_j_int
4001         }
4002         { \bool_set_true:N \l_@@_stop_loop_bool }
4003         {
4004             \cs_set:cpn
4005             {
4006                 @@ _ dotted _
4007                 \int_use:N \l_@@_initial_i_int -
4008                 \int_use:N \l_@@_initial_j_int
4009             }
4010             { }
4011         }
4012     }
4013 }
4014 }

```

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.

```

4015     \seq_gput_right:Nx \g_@@_pos_of_xdots_seq
4016     {
4017         { \int_use:N \l_@@_initial_i_int }

```

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

```

4018         { \int_min:nn \l_@@_initial_j_int \l_@@_final_j_int }
4019         { \int_use:N \l_@@_final_i_int }
4020         { \int_max:nn \l_@@_initial_j_int \l_@@_final_j_int }
4021         { } % for the name of the block
4022     }
4023 }

```

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 known wheter the extremities are closed or open) but before the analyse of the

keys of the individual command `\Cdots`, `\Vdots`. Hence, the keys `shorten`, `shorten-start` and `shorten-end` of that individual command will be applied.

```

4024 \cs_new_protected:Npn \@@_open_shorten:
4025 {
4026   \bool_if:NT \l_@@_initial_open_bool
4027     { \dim_zero:N \l_@@_xdots_shorten_start_dim }
4028   \bool_if:NT \l_@@_final_open_bool
4029     { \dim_zero:N \l_@@_xdots_shorten_end_dim }
4030 }

```

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).

```

4031 \cs_new_protected:Npn \@@_adjust_to_submatrix:nn #1 #2
4032 {
4033   \int_set:Nn \l_@@_row_min_int 1
4034   \int_set:Nn \l_@@_col_min_int 1
4035   \int_set_eq:NN \l_@@_row_max_int \c@iRow
4036   \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`.

```

4037   \seq_map_inline:Nn \g_@@_submatrix_seq
4038     { \@@_adjust_to_submatrix:nnnnnn { #1 } { #2 } ##1 }
4039 }

```

#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.

```

4040 \cs_set_protected:Npn \@@_adjust_to_submatrix:nnnnnn #1 #2 #3 #4 #5 #6
4041 {
4042   \bool_if:nT
4043     {
4044       \int_compare_p:n { #3 <= #1 }
4045       && \int_compare_p:n { #1 <= #5 }
4046       && \int_compare_p:n { #4 <= #2 }
4047       && \int_compare_p:n { #2 <= #6 }
4048     }
4049     {
4050       \int_set:Nn \l_@@_row_min_int { \int_max:nn \l_@@_row_min_int { #3 } }
4051       \int_set:Nn \l_@@_col_min_int { \int_max:nn \l_@@_col_min_int { #4 } }
4052       \int_set:Nn \l_@@_row_max_int { \int_min:nn \l_@@_row_max_int { #5 } }
4053       \int_set:Nn \l_@@_col_max_int { \int_min:nn \l_@@_col_max_int { #6 } }
4054     }
4055 }

```

```

4056 \cs_new_protected:Npn \@@_set_initial_coords:
4057 {
4058   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4059   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4060 }
4061 \cs_new_protected:Npn \@@_set_final_coords:
4062 {
4063   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4064   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4065 }
4066 \cs_new_protected:Npn \@@_set_initial_coords_from_anchor:n #1
4067 {
4068   \pgfpointanchor
4069     {
4070       \@@_env:
4071       - \int_use:N \l_@@_initial_i_int

```

```

4072     - \int_use:N \l_@@_initial_j_int
4073   }
4074   { #1 }
4075   \@@_set_initial_coords:
4076 }
4077 \cs_new_protected:Npn \@@_set_final_coords_from_anchor:n #1
4078 {
4079   \pgfpointanchor
4080   {
4081     \@@_env:
4082     - \int_use:N \l_@@_final_i_int
4083     - \int_use:N \l_@@_final_j_int
4084   }
4085   { #1 }
4086   \@@_set_final_coords:
4087 }
4088 \cs_new_protected:Npn \@@_open_x_initial_dim:
4089 {
4090   \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
4091   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
4092   {
4093     \cs_if_exist:cT
4094     { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4095     {
4096       \pgfpointanchor
4097       { \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4098       { west }
4099       \dim_set:Nn \l_@@_x_initial_dim
4100       { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
4101     }
4102   }

```

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

```

4103   \dim_compare:nNnT \l_@@_x_initial_dim = \c_max_dim
4104   {
4105     \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4106     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4107     \dim_add:Nn \l_@@_x_initial_dim \col@sep
4108   }
4109 }
4110 \cs_new_protected:Npn \@@_open_x_final_dim:
4111 {
4112   \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
4113   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
4114   {
4115     \cs_if_exist:cT
4116     { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4117     {
4118       \pgfpointanchor
4119       { \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4120       { east }
4121       \dim_set:Nn \l_@@_x_final_dim
4122       { \dim_max:nn \l_@@_x_final_dim \pgf@x }
4123     }
4124   }

```

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

```

4125   \dim_compare:nNnT \l_@@_x_final_dim = { - \c_max_dim }
4126   {
4127     \@@_qpoint:n { col - \int_eval:n { \l_@@_final_j_int + 1 } }
4128     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4129     \dim_sub:Nn \l_@@_x_final_dim \col@sep
4130   }
4131 }

```

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.

```

4132 \cs_new_protected:Npn \@@_draw_Ldots:nnn #1 #2 #3
4133 {
4134   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4135   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4136   {
4137     \@@_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.

```

4138     \group_begin:
4139     \@@_open_shorten:
4140     \int_compare:nNnTF { #1 } = 0
4141     { \color { nicematrix-first-row } }
4142     {

```

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.

```

4143         \int_compare:nNnT { #1 } = \l_@@_last_row_int
4144         { \color { nicematrix-last-row } }
4145     }
4146     \keys_set:nn { NiceMatrix / xdots } { #3 }
4147     \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4148     \@@_actually_draw_Ldots:
4149     \group_end:
4150   }
4151 }

```

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`.

```

4152 \cs_new_protected:Npn \@@_actually_draw_Ldots:
4153 {
4154   \bool_if:NTF \l_@@_initial_open_bool
4155   {
4156     \@@_open_x_initial_dim:
4157     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4158     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4159   }
4160   { \@@_set_initial_coords_from_anchor:n { base-east } }
4161   \bool_if:NTF \l_@@_final_open_bool
4162   {
4163     \@@_open_x_final_dim:
4164     \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4165     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4166   }
4167   { \@@_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.

```

4168   \bool_lazy_all:nTF

```

```

4169 {
4170   \l_@@_initial_open_bool
4171   \l_@@_final_open_bool
4172   { \int_compare_p:nNn \l_@@_initial_i_int = \l_@@_last_row_int }
4173 }
4174 {
4175   \dim_add:Nn \l_@@_y_initial_dim \c_@@_shift_Ldots_last_row_dim
4176   \dim_add:Nn \l_@@_y_final_dim \c_@@_shift_Ldots_last_row_dim
4177 }

```

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

```

4178 {
4179   \dim_add:Nn \l_@@_y_initial_dim \l_@@_xdots_radius_dim
4180   \dim_add:Nn \l_@@_y_final_dim \l_@@_xdots_radius_dim
4181 }
4182 \@@_draw_line:
4183 }

```

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.

```

4184 \cs_new_protected:Npn \@@_draw_Cdots:nnn #1 #2 #3
4185 {
4186   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4187   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4188   {
4189     \@@_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.

```

4190   \group_begin:
4191   \@@_open_shorten:
4192   \int_compare:nNnTF { #1 } = 0
4193     { \color { nicematrix-first-row } }
4194   {

```

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.

```

4195     \int_compare:nNnT { #1 } = \l_@@_last_row_int
4196     { \color { nicematrix-last-row } }
4197   }
4198   \keys_set:nn { NiceMatrix / xdots } { #3 }
4199   \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4200   \@@_actually_draw_Cdots:
4201   \group_end:
4202 }
4203 }

```

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`.

```

4204 \cs_new_protected:Npn \@@_actually_draw_Cdots:
4205 {
4206   \bool_if:NTF \l_@@_initial_open_bool
4207     { \@@_open_x_initial_dim: }
4208     { \@@_set_initial_coords_from_anchor:n { mid-east } }
4209   \bool_if:NTF \l_@@_final_open_bool
4210     { \@@_open_x_final_dim: }
4211     { \@@_set_final_coords_from_anchor:n { mid-west } }
4212   \bool_lazy_and:nnTF
4213     \l_@@_initial_open_bool
4214     \l_@@_final_open_bool
4215     {
4216       \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int }
4217       \dim_set_eq:NN \l_tmpa_dim \pgf@y
4218       \@@_qpoint:n { row - \int_eval:n { \l_@@_initial_i_int + 1 } }
4219       \dim_set:Nn \l_@@_y_initial_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
4220       \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
4221     }
4222     {
4223       \bool_if:NT \l_@@_initial_open_bool
4224         { \dim_set_eq:NN \l_@@_y_initial_dim \l_@@_y_final_dim }
4225       \bool_if:NT \l_@@_final_open_bool
4226         { \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim }
4227     }
4228   \@@_draw_line:
4229 }
4230 \cs_new_protected:Npn \@@_open_y_initial_dim:
4231 {
4232   \dim_set:Nn \l_@@_y_initial_dim { - \c_max_dim }
4233   \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
4234   {
4235     \cs_if_exist:cT
4236       { pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4237       {
4238         \pgfpointanchor
4239           { \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4240           { north }
4241         \dim_set:Nn \l_@@_y_initial_dim
4242           { \dim_max:nn \l_@@_y_initial_dim \pgf@y }
4243       }
4244   }
4245   % modified 2023-08-10
4246   \dim_compare:nNnT \l_@@_y_initial_dim = { - \c_max_dim }
4247   {
4248     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4249     \dim_set:Nn \l_@@_y_initial_dim
4250     {
4251       \fp_to_dim:n
4252       {
4253         \pgf@y
4254         + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
4255       }
4256     }
4257   }
4258 }
4259 \cs_new_protected:Npn \@@_open_y_final_dim:
4260 {
4261   \dim_set_eq:NN \l_@@_y_final_dim \c_max_dim
4262   \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
4263   {
4264     \cs_if_exist:cT
4265       { pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4266       {

```



```

4267         \pgfpointanchor
4268         { \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4269         { south }
4270         \dim_set:Nn \l_@@_y_final_dim
4271         { \dim_min:nn \l_@@_y_final_dim \pgf@y }
4272     }
4273 }
4274 % modified 2023-08-10
4275 \dim_compare:nNnT \l_@@_y_final_dim = \c_max_dim
4276 {
4277     \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4278     \dim_set:Nn \l_@@_y_final_dim
4279     { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
4280 }
4281 }

```

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.

```

4282 \cs_new_protected:Npn \@@_draw_Vdots:nnn #1 #2 #3
4283 {
4284     \@@_adjust_to_submatrix:nn { #1 } { #2 }
4285     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4286     {
4287         \@@_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.

```

4288     \group_begin:
4289     \@@_open_shorten:
4290     \int_compare:nNnTF { #2 } = 0
4291     { \color { nicematrix-first-col } }
4292     {
4293         \int_compare:nNnT { #2 } = \l_@@_last_col_int
4294         { \color { nicematrix-last-col } }
4295     }
4296     \keys_set:nn { NiceMatrix / xdots } { #3 }
4297     \tl_if_empty:VF \l_@@_xdots_color_tl
4298     { \color { \l_@@_xdots_color_tl } }
4299     \@@_actually_draw_Vdots:
4300     \group_end:
4301 }
4302 }

```

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`.

```

4303 \cs_new_protected:Npn \@@_actually_draw_Vdots:
4304 {

```

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

```

4305     \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.

```

4306 {
4307   \@@_open_y_initial_dim:
4308   \@@_open_y_final_dim:
4309   \int_compare:nNnTF \l_@@_initial_j_int = \c_zero_int

```

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

```

4310 {
4311   \@@_qpoint:n { col - 1 }
4312   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4313   \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
4314   \dim_sub:Nn \l_@@_x_initial_dim \l_@@_extra_left_margin_dim
4315   % \bool_if:NT \g_@@_delims_bool
4316   % {
4317     \dim_sub:Nn \l_@@_x_initial_dim \c_@@_shift_exterior_Vdots_dim
4318   % }
4319 }
4320 {
4321   \bool_lazy_and:nnTF
4322   { \int_compare_p:nNn \l_@@_last_col_int > { -2 } }
4323   { \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”.

```

4324 {
4325   \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4326   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4327   \dim_add:Nn \l_@@_x_initial_dim \l_@@_right_margin_dim
4328   \dim_add:Nn \l_@@_x_initial_dim \l_@@_extra_right_margin_dim
4329   % \bool_if:NT \g_@@_delims_bool
4330   % {
4331     \dim_add:Nn
4332     \l_@@_x_initial_dim
4333     \c_@@_shift_exterior_Vdots_dim
4334   % }
4335 }

```

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

```

4336 {
4337   \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4338   \dim_set_eq:NN \l_tmpa_dim \pgf@x
4339   \@@_qpoint:n { col - \int_eval:n { \l_@@_initial_j_int + 1 } }
4340   \dim_set:Nn \l_@@_x_initial_dim { ( \pgf@x + \l_tmpa_dim ) / 2 }
4341 }
4342 }
4343 }

```

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.

```

4344 {
4345   \bool_set_false:N \l_tmpa_bool
4346   \bool_lazy_and:nnT
4347   { ! \l_@@_initial_open_bool }
4348   { ! \l_@@_final_open_bool }
4349   {
4350     \@@_set_initial_coords_from_anchor:n { south-west }
4351     \@@_set_final_coords_from_anchor:n { north-west }
4352     \bool_set:Nn \l_tmpa_bool
4353     { \dim_compare_p:nNn \l_@@_x_initial_dim = \l_@@_x_final_dim }
4354   }

```

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

```

4355   \bool_if:NTF \l_@@_initial_open_bool
4356   {
4357     \@@_open_y_initial_dim:
4358     \@@_set_final_coords_from_anchor:n { north }

```

```

4359         \dim_set_eq:NN \l_@@_x_initial_dim \l_@@_x_final_dim
4360     }
4361     {
4362         \@@_set_initial_coords_from_anchor:n { south }
4363         \bool_if:NTF \l_@@_final_open_bool
4364             \@@_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.

```

4365         {
4366             \@@_set_final_coords_from_anchor:n { north }
4367             \dim_compare:nNnF \l_@@_x_initial_dim = \l_@@_x_final_dim
4368             {
4369                 \dim_set:Nn \l_@@_x_initial_dim
4370                 {
4371                     \bool_if:NTF \l_tmpa_bool \dim_min:nn \dim_max:nn
4372                     \l_@@_x_initial_dim \l_@@_x_final_dim
4373                 }
4374             }
4375         }
4376     }
4377 }
4378 \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
4379 \@@_draw_line:
4380 }

```

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.

```

4381 \cs_new_protected:Npn \@@_draw_Ddots:nnn #1 #2 #3
4382 {
4383     \@@_adjust_to_submatrix:nn { #1 } { #2 }
4384     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4385     {
4386         \@@_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.

```

4387     \group_begin:
4388     \@@_open_shorten:
4389     \keys_set:nn { NiceMatrix / xdots } { #3 }
4390     \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4391     \@@_actually_draw_Ddots:
4392     \group_end:
4393 }
4394 }

```

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`.

```

4395 \cs_new_protected:Npn \@@_actually_draw_Ddots:
4396 {
4397   \bool_if:NTF \l_@@_initial_open_bool
4398   {
4399     \@@_open_y_initial_dim:
4400     \@@_open_x_initial_dim:
4401   }
4402   { \@@_set_initial_coords_from_anchor:n { south-east } }
4403   \bool_if:NTF \l_@@_final_open_bool
4404   {
4405     \@@_open_x_final_dim:
4406     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4407   }
4408   { \@@_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.

```

4409   \bool_if:NT \l_@@_parallelize_diags_bool
4410   {
4411     \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).

```

4412     \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.

```

4413     {
4414       \dim_gset:Nn \g_@@_delta_x_one_dim
4415       { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4416       \dim_gset:Nn \g_@@_delta_y_one_dim
4417       { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4418     }

```

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`.

```

4419     {
4420       \dim_set:Nn \l_@@_y_final_dim
4421       {
4422         \l_@@_y_initial_dim +
4423         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4424         \dim_ratio:nn \g_@@_delta_y_one_dim \g_@@_delta_x_one_dim
4425       }
4426     }
4427   }
4428   \@@_draw_line:
4429 }

```

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.

```

4430 \cs_new_protected:Npn \@@_draw_Iddots:nnn #1 #2 #3
4431 {
4432   \@@_adjust_to_submatrix:nn { #1 } { #2 }
4433   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4434   {
4435     \@@_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.

```

4436     \group_begin:
4437     \@@_open_shorten:
4438     \keys_set:nn { NiceMatrix / xdots } { #3 }
4439     \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }

```

```

4440         \@@_actually_draw_Iddots:
4441     \group_end:
4442 }
4443 }

```

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.`

```

4444 \cs_new_protected:Npn \@@_actually_draw_Iddots:
4445 {
4446     \bool_if:NTF \l_@@_initial_open_bool
4447     {
4448         \@@_open_y_initial_dim:
4449         \@@_open_x_initial_dim:
4450     }
4451     { \@@_set_initial_coords_from_anchor:n { south-west } }
4452     \bool_if:NTF \l_@@_final_open_bool
4453     {
4454         \@@_open_y_final_dim:
4455         \@@_open_x_final_dim:
4456     }
4457     { \@@_set_final_coords_from_anchor:n { north-east } }
4458     \bool_if:NT \l_@@_parallelize_diags_bool
4459     {
4460         \int_gincr:N \g_@@_iddots_int
4461         \int_compare:nNnTF \g_@@_iddots_int = 1
4462         {
4463             \dim_gset:Nn \g_@@_delta_x_two_dim
4464             { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4465             \dim_gset:Nn \g_@@_delta_y_two_dim
4466             { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4467         }
4468         {
4469             \dim_set:Nn \l_@@_y_final_dim
4470             {
4471                 \l_@@_y_initial_dim +
4472                 ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4473                 \dim_ratio:nn \g_@@_delta_y_two_dim \g_@@_delta_x_two_dim
4474             }
4475         }
4476     }
4477     \@@_draw_line:
4478 }

```

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`

```

4479 \cs_new_protected:Npn \@@_draw_line:
4480 {
4481   \pgfrememberpicturepositiononpagetrue
4482   \pgf@relevantforpicturesizefalse
4483   \bool_lazy_or:nnTF
4484     { \tl_if_eq_p:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl }
4485     \l_@@_dotted_bool
4486     \@@_draw_standard_dotted_line:
4487     \@@_draw_unstandard_dotted_line:
4488 }

```

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.

```

4489 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:
4490 {
4491   \begin { scope }
4492   \@@_draw_unstandard_dotted_line:o
4493     { \l_@@_xdots_line_style_tl , \l_@@_xdots_color_tl }
4494 }

```

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.

```

4495 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:n #1
4496 {
4497   \@@_draw_unstandard_dotted_line:nVVV
4498     { #1 }
4499     \l_@@_xdots_up_tl
4500     \l_@@_xdots_down_tl
4501     \l_@@_xdots_middle_tl
4502 }
4503 \cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:n { o }

```

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

```

4504 \hook_gput_code:nnn { begindocument } { . }
4505 {
4506   \IfPackageLoadedTF { tikz }
4507   {
4508     \tikzset
4509     {
4510       @@_node_above / .style = { sloped , above } ,
4511       @@_node_below / .style = { sloped , below } ,
4512       @@_node_middle / .style =
4513       {
4514         sloped ,
4515         inner~sep = \c_@@_innersep_middle_dim
4516       }
4517     }
4518   }
4519   { }
4520 }

```

```

4521 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:nnnn #1 #2 #3 #4
4522 {

```

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.

```

4523 \dim_zero_new:N \l_@@_l_dim
4524 \dim_set:Nn \l_@@_l_dim
4525 {
4526   \fp_to_dim:n
4527   {
4528     sqrt
4529     (
4530       ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
4531       +
4532       ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
4533     )
4534   }
4535 }
4536 \bool_lazy_and:nnT % security
4537 { \dim_compare_p:nNn { \dim_abs:n \l_@@_l_dim } < \c_@@_max_l_dim }
4538 { \dim_compare_p:nNn { \dim_abs:n \l_@@_l_dim } > { 1 pt } }
4539 {
4540   \dim_set:Nn \l_tmpa_dim
4541   {
4542     \l_@@_x_initial_dim
4543     + ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
4544     * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
4545   }
4546   \dim_set:Nn \l_tmpb_dim
4547   {
4548     \l_@@_y_initial_dim
4549     + ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
4550     * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
4551   }
4552   \dim_set:Nn \l_@@_tmpc_dim
4553   {
4554     \l_@@_x_final_dim
4555     - ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
4556     * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
4557   }
4558   \dim_set:Nn \l_@@_tmpd_dim
4559   {
4560     \l_@@_y_final_dim
4561     - ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
4562     * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
4563   }
4564   \dim_set_eq:NN \l_@@_x_initial_dim \l_tmpa_dim
4565   \dim_set_eq:NN \l_@@_y_initial_dim \l_tmpb_dim
4566   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_tmpc_dim
4567   \dim_set_eq:NN \l_@@_y_final_dim \l_@@_tmpd_dim
4568 }

```

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

```

4569 \bool_if:NT \l_@@_xdots_h_labels_bool
4570 {
4571   \tikzset
4572   {
4573     @@_node_above / .style = { auto = left } ,
4574     @@_node_below / .style = { auto = right } ,
4575     @@_node_middle / .style = { inner~sep = \c_@@_innersep_middle_dim }

```

```

4576     }
4577   }
4578   \tl_if_empty:nF { #4 }
4579   { \tikzset { @@_node_middle / .append~style = { fill = white } } }
4580   \draw
4581     [ #1 ]
4582     ( \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).

```

4583     -- node [ @@_node_middle ] { $ \scriptstyle #4 $ }
4584     node [ @@_node_below ] { $ \scriptstyle #3 $ }
4585     node [ @@_node_above ] { $ \scriptstyle #2 $ }
4586     ( \l_@@_x_final_dim , \l_@@_y_final_dim ) ;
4587   \end { scope }
4588 }
4589 \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).

```

4590 \cs_new_protected:Npn \@@_draw_standard_dotted_line:
4591 {
4592   \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.

```

4593   \dim_zero_new:N \l_@@_l_dim
4594   \dim_set:Nn \l_@@_l_dim
4595   {
4596     \fp_to_dim:n
4597     {
4598       sqrt
4599       (
4600         ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
4601         +
4602         ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
4603       )
4604     }
4605   }

```

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.

```

4606   \bool_lazy_or:nnF
4607   { \dim_compare_p:nNn { \dim_abs:n \l_@@_l_dim } > \c_@@_max_l_dim }
4608   { \dim_compare_p:nNn \l_@@_l_dim = \c_zero_dim }
4609   \@@_draw_standard_dotted_line_i:
4610 \group_end:
4611 \bool_lazy_all:nF
4612 {
4613   { \tl_if_empty_p:N \l_@@_xdots_up_tl }
4614   { \tl_if_empty_p:N \l_@@_xdots_down_tl }
4615   { \tl_if_empty_p:N \l_@@_xdots_middle_tl }
4616 }
4617 \l_@@_labels_standard_dotted_line:
4618 }
4619 \dim_const:Nn \c_@@_max_l_dim { 50 cm }
4620 \cs_new_protected:Npn \@@_draw_standard_dotted_line_i:
4621 {

```


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

```

4622 \int_set:Nn \l_tmpa_int
4623 {
4624   \dim_ratio:nn
4625   {
4626     \l_@@_l_dim
4627     - \l_@@_xdots_shorten_start_dim
4628     - \l_@@_xdots_shorten_end_dim
4629   }
4630   \l_@@_xdots_inter_dim
4631 }

```

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

```

4632 \dim_set:Nn \l_tmpa_dim
4633 {
4634   ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4635   \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
4636 }
4637 \dim_set:Nn \l_tmpb_dim
4638 {
4639   ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
4640   \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
4641 }

```

In the loop over the dots, the dimensions $\backslash l_@@_x_initial_dim$ and $\backslash l_@@_y_initial_dim$ will be used for the coordinates of the dots. But, before the loop, we must move until the first dot.

```

4642 \dim_gadd:Nn \l_@@_x_initial_dim
4643 {
4644   ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4645   \dim_ratio:nn
4646   {
4647     \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
4648     + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
4649   }
4650   { 2 \l_@@_l_dim }
4651 }
4652 \dim_gadd:Nn \l_@@_y_initial_dim
4653 {
4654   ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
4655   \dim_ratio:nn
4656   {
4657     \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
4658     + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
4659   }
4660   { 2 \l_@@_l_dim }
4661 }
4662 \pgf@relevantforpicturesizefalse
4663 \int_step_inline:nnn 0 \l_tmpa_int
4664 {
4665   \pgfpathcircle
4666   { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
4667   { \l_@@_xdots_radius_dim }
4668   \dim_add:Nn \l_@@_x_initial_dim \l_tmpa_dim
4669   \dim_add:Nn \l_@@_y_initial_dim \l_tmpb_dim
4670 }
4671 \pgfusepathqfill
4672 }

4673 \cs_new_protected:Npn \l_@@_labels_standard_dotted_line:
4674 {
4675   \pgfscope

```

```

4676 \pgftransformshift
4677 {
4678   \pgfpointlineatime { 0.5 }
4679   { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
4680   { \pgfpoint \l_@@_x_final_dim \l_@@_y_final_dim }
4681 }
4682 \fp_set:Nn \l_tmpa_fp
4683 {
4684   atand
4685   (
4686     \l_@@_y_final_dim - \l_@@_y_initial_dim ,
4687     \l_@@_x_final_dim - \l_@@_x_initial_dim
4688   )
4689 }
4690 \pgftransformrotate { \fp_use:N \l_tmpa_fp }
4691 \bool_if:NF \l_@@_xdots_h_labels_bool { \fp_zero:N \l_tmpa_fp }
4692 \tl_if_empty:NF \l_@@_xdots_middle_tl
4693 {
4694   \begin { pgfscope }
4695   \pgfset { inner~sep = \c_@@_innersep_middle_dim }
4696   \pgfnode
4697   { rectangle }
4698   { center }
4699   {
4700     \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4701     {
4702       \c_math_toggle_token
4703       \scriptstyle \l_@@_xdots_middle_tl
4704       \c_math_toggle_token
4705     }
4706   }
4707   { }
4708   {
4709     \pgfsetfillcolor { white }
4710     \pgfusepath { fill }
4711   }
4712   \end { pgfscope }
4713 }
4714 \tl_if_empty:NF \l_@@_xdots_up_tl
4715 {
4716   \pgfnode
4717   { rectangle }
4718   { south }
4719   {
4720     \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4721     {
4722       \c_math_toggle_token
4723       \scriptstyle \l_@@_xdots_up_tl
4724       \c_math_toggle_token
4725     }
4726   }
4727   { }
4728   { \pgfusepath { } }
4729 }
4730 \tl_if_empty:NF \l_@@_xdots_down_tl
4731 {
4732   \pgfnode
4733   { rectangle }
4734   { north }
4735   {
4736     \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4737     {
4738       \c_math_toggle_token

```

```

4739         \scriptstyle \l_@@_xdots_down_tl
4740         \c_math_toggle_token
4741     }
4742 }
4743 { }
4744 { \pgfusepath { } }
4745 }
4746 \endpgfscope
4747 }

```

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.

```

4748 \hook_gput_code:nnn { begindocument } { . }
4749 {
4750     \tl_set:Nn \l_@@_argspec_tl { m E { _ ^ : } { { } { } { } } }
4751     \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
4752     \cs_new_protected:Npn \@@_Ldots
4753     { \@@_collect_options:n { \@@_Ldots_i } }
4754     \exp_args:NNV \NewDocumentCommand \@@_Ldots_i \l_@@_argspec_tl
4755     {
4756         \int_compare:nNnTF \c@jCol = 0
4757         { \@@_error:nn { in~first~col } \Ldots }
4758         {
4759             \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
4760             { \@@_error:nn { in~last~col } \Ldots }
4761             {
4762                 \@@_instruction_of_type:nnn \c_false_bool { Ldots }
4763                 { #1 , down = #2 , up = #3 , middle = #4 }
4764             }
4765         }
4766         \bool_if:NF \l_@@_nullify_dots_bool
4767         { \phantom { \ensuremath { \@@_old_ldots } } }
4768         \bool_gset_true:N \g_@@_empty_cell_bool
4769     }

4770     \cs_new_protected:Npn \@@_Cdots
4771     { \@@_collect_options:n { \@@_Cdots_i } }
4772     \exp_args:NNV \NewDocumentCommand \@@_Cdots_i \l_@@_argspec_tl
4773     {
4774         \int_compare:nNnTF \c@jCol = 0
4775         { \@@_error:nn { in~first~col } \Cdots }
4776         {
4777             \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
4778             { \@@_error:nn { in~last~col } \Cdots }
4779             {
4780                 \@@_instruction_of_type:nnn \c_false_bool { Cdots }
4781                 { #1 , down = #2 , up = #3 , middle = #4 }
4782             }
4783         }
4784     }

```

```

4784 \bool_if:NF \l_@@_nullify_dots_bool
4785 { \phantom { \ensuremath { \@@_old_cdots } } }
4786 \bool_gset_true:N \g_@@_empty_cell_bool
4787 }

4788 \cs_new_protected:Npn \@@_Vdots
4789 { \@@_collect_options:n { \@@_Vdots_i } }
4790 \exp_args:NNV \NewDocumentCommand \@@_Vdots_i \l_@@_argspec_tl
4791 {
4792   \int_compare:nNnTF \c@iRow = 0
4793   { \@@_error:nn { in~first~row } \Vdots }
4794   {
4795     \int_compare:nNnTF \c@iRow = \l_@@_last_row_int
4796     { \@@_error:nn { in~last~row } \Vdots }
4797     {
4798       \@@_instruction_of_type:nnn \c_false_bool { \Vdots }
4799       { #1 , down = #2 , up = #3 , middle = #4 }
4800     }
4801   }
4802   \bool_if:NF \l_@@_nullify_dots_bool
4803   { \phantom { \ensuremath { \@@_old_vdots } } }
4804   \bool_gset_true:N \g_@@_empty_cell_bool
4805 }

4806 \cs_new_protected:Npn \@@_Ddots
4807 { \@@_collect_options:n { \@@_Ddots_i } }
4808 \exp_args:NNV \NewDocumentCommand \@@_Ddots_i \l_@@_argspec_tl
4809 {
4810   \int_case:nnF \c@iRow
4811   {
4812     0 { \@@_error:nn { in~first~row } \Ddots }
4813     \l_@@_last_row_int { \@@_error:nn { in~last~row } \Ddots }
4814   }
4815   {
4816     \int_case:nnF \c@jCol
4817     {
4818       0 { \@@_error:nn { in~first~col } \Ddots }
4819       \l_@@_last_col_int { \@@_error:nn { in~last~col } \Ddots }
4820     }
4821     {
4822       \keys_set_known:nn { NiceMatrix / Ddots } { #1 }
4823       \@@_instruction_of_type:nnn \l_@@_draw_first_bool { \Ddots }
4824       { #1 , down = #2 , up = #3 , middle = #4 }
4825     }
4826   }
4827 }
4828 \bool_if:NF \l_@@_nullify_dots_bool
4829 { \phantom { \ensuremath { \@@_old_ddots } } }
4830 \bool_gset_true:N \g_@@_empty_cell_bool
4831 }

4832 \cs_new_protected:Npn \@@_Iddots
4833 { \@@_collect_options:n { \@@_Iddots_i } }
4834 \exp_args:NNV \NewDocumentCommand \@@_Iddots_i \l_@@_argspec_tl
4835 {
4836   \int_case:nnF \c@iRow
4837   {
4838     0 { \@@_error:nn { in~first~row } \Iddots }
4839     \l_@@_last_row_int { \@@_error:nn { in~last~row } \Iddots }
4840   }
4841   {

```

```

4842 \int_case:nnF \c@jCol
4843 {
4844     0 { \@@_error:nn { in~first~col } \Iddots }
4845     \l_@@_last_col_int { \@@_error:nn { in~last~col } \Iddots }
4846 }
4847 {
4848     \keys_set_known:nn { NiceMatrix / Ddots } { #1 }
4849     \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Iddots }
4850     { #1 , down = #2 , up = #3 , middle = #4 }
4851 }
4852 }
4853 \bool_if:NF \l_@@_nullify_dots_bool
4854 { \phantom { \ensuremath { \@@_old_iddots } } }
4855 \bool_gset_true:N \g_@@_empty_cell_bool
4856 }
4857 }

```

End of the \AddToHook.

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

```

4858 \keys_define:nn { NiceMatrix / Ddots }
4859 {
4860     draw-first .bool_set:N = \l_@@_draw_first_bool ,
4861     draw-first .default:n = true ,
4862     draw-first .value_forbidden:n = true
4863 }

```

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

```

4864 \cs_new_protected:Npn \@@_Hspace:
4865 {
4866     \bool_gset_true:N \g_@@_empty_cell_bool
4867     \hspace
4868 }

```

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`.

```

4869 \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`.

```

4870 \cs_new:Npn \@@_Hdotsfor:
4871 {
4872     \bool_lazy_and:nnTF
4873     { \int_compare_p:nNn \c@jCol = 0 }
4874     { \int_compare_p:nNn \l_@@_first_col_int = 0 }
4875     {
4876         \bool_if:NTF \g_@@_after_col_zero_bool
4877         {
4878             \multicolumn { 1 } { c } { }
4879             \@@_Hdotsfor_i
4880         }
4881         { \@@_fatal:n { Hdotsfor~in~col~0 } }
4882     }
4883     {
4884         \multicolumn { 1 } { c } { }
4885         \@@_Hdotsfor_i
4886     }
4887 }

```

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:`).

```

4888 \hook_gput_code:nnn { begindocument } { . }
4889 {
4890   \tl_set:Nn \l_@@_argspec_tl { m m 0 { } E { _ ^ : } { { } { } { } } }
4891   \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.

```

4892   \cs_new_protected:Npn \@@_Hdotsfor_i
4893     { \@@_collect_options:n { \@@_Hdotsfor_ii } }
4894   \exp_args:NNV \NewDocumentCommand \@@_Hdotsfor_ii \l_@@_argspec_tl
4895     {
4896       \tl_gput_right:Nx \g_@@_HVDotsfor_lines_tl
4897       {
4898         \@@_Hdotsfor:nnnn
4899         { \int_use:N \c@iRow }
4900         { \int_use:N \c@jCol }
4901         { #2 }
4902         {
4903           #1 , #3 ,
4904           down = \exp_not:n { #4 } ,
4905           up = \exp_not:n { #5 } ,
4906           middle = \exp_not:n { #6 }
4907         }
4908       }
4909       \prg_replicate:nn { #2 - 1 }
4910       {
4911         &
4912         \multicolumn { 1 } { c } { }
4913         \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i: % added 2023-08-26
4914       }
4915     }
4916   }

```

```

4917 \cs_new_protected:Npn \@@_Hdotsfor:nnnn #1 #2 #3 #4
4918 {
4919   \bool_set_false:N \l_@@_initial_open_bool
4920   \bool_set_false:N \l_@@_final_open_bool

```

For the row, it's easy.

```

4921   \int_set:Nn \l_@@_initial_i_int { #1 }
4922   \int_set_eq:NN \l_@@_final_i_int \l_@@_initial_i_int

```

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

```

4923   \int_compare:nNnTF { #2 } = 1
4924   {
4925     \int_set:Nn \l_@@_initial_j_int 1
4926     \bool_set_true:N \l_@@_initial_open_bool
4927   }
4928   {
4929     \cs_if_exist:cTF
4930     {
4931       pgf @ sh @ ns @ \@@_env:
4932       - \int_use:N \l_@@_initial_i_int
4933       - \int_eval:n { #2 - 1 }
4934     }
4935     { \int_set:Nn \l_@@_initial_j_int { #2 - 1 } }
4936     {
4937       \int_set:Nn \l_@@_initial_j_int { #2 }
4938       \bool_set_true:N \l_@@_initial_open_bool
4939     }
4940   }

```

```

4941 \int_compare:nNnTF { #2 + #3 - 1 } = \c@jCol
4942 {
4943   \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
4944   \bool_set_true:N \l_@@_final_open_bool
4945 }
4946 {
4947   \cs_if_exist:cTF
4948   {
4949     pgf @ sh @ ns @ \@@_env:
4950     - \int_use:N \l_@@_final_i_int
4951     - \int_eval:n { #2 + #3 }
4952   }
4953   { \int_set:Nn \l_@@_final_j_int { #2 + #3 } }
4954   {
4955     \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
4956     \bool_set_true:N \l_@@_final_open_bool
4957   }
4958 }
4959 \group_begin:
4960
4961 \@@_open_shorten:
4962
4963
4964 \int_compare:nNnTF { #1 } = 0
4965 { \color { nicematrix-first-row } }
4966 {
4967   \int_compare:nNnT { #1 } = \g_@@_row_total_int
4968   { \color { nicematrix-last-row } }
4969 }
4970
4971 \keys_set:nn { NiceMatrix / xdots } { #4 }
4972 \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4973 \@@_actually_draw_Ldots:
4974 \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).

```

4975 \int_step_inline:nnn { #2 } { #2 + #3 - 1 }
4976 { \cs_set:cpn { @@ _ dotted _ #1 - ##1 } { } }
4977 }

4978 \hook_gput_code:nnn { begindocument } { . }
4979 {
4980   \tl_set:Nn \l_@@_argspec_tl { m m 0 { } E { _ ^ : } { { } { } { } } }
4981   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
4982   \cs_new_protected:Npn \@@_Vdotsfor:
4983   { \@@_collect_options:n { \@@_Vdotsfor_i } }
4984   \exp_args:NNV \NewDocumentCommand \@@_Vdotsfor_i \l_@@_argspec_tl
4985   {
4986     \bool_gset_true:N \g_@@_empty_cell_bool
4987     \tl_gput_right:Nx \g_@@_HVdotsfor_lines_tl
4988     {
4989       \@@_Vdotsfor:nnnn
4990       { \int_use:N \c@iRow }
4991       { \int_use:N \c@jCol }
4992       { #2 }
4993       {
4994         #1 , #3 ,
4995         down = \exp_not:n { #4 } ,
4996         up = \exp_not:n { #5 } ,
4997         middle = \exp_not:n { #6 }

```

```

4998     }
4999   }
5000 }
5001 }

```

```

5002 \cs_new_protected:Npn \@@_Vdotsfor:nnnn #1 #2 #3 #4
5003 {
5004   \bool_set_false:N \l_@@_initial_open_bool
5005   \bool_set_false:N \l_@@_final_open_bool

```

For the column, it's easy.

```

5006   \int_set:Nn \l_@@_initial_j_int { #2 }
5007   \int_set_eq:NN \l_@@_final_j_int \l_@@_initial_j_int

```

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

```

5008   \int_compare:nNnTF { #1 } = 1
5009   {
5010     \int_set:Nn \l_@@_initial_i_int 1
5011     \bool_set_true:N \l_@@_initial_open_bool
5012   }
5013   {
5014     \cs_if_exist:cTF
5015     {
5016       pgf @ sh @ ns @ \@@_env:
5017       - \int_eval:n { #1 - 1 }
5018       - \int_use:N \l_@@_initial_j_int
5019     }
5020     { \int_set:Nn \l_@@_initial_i_int { #1 - 1 } }
5021     {
5022       \int_set:Nn \l_@@_initial_i_int { #1 }
5023       \bool_set_true:N \l_@@_initial_open_bool
5024     }
5025   }
5026   \int_compare:nNnTF { #1 + #3 - 1 } = \c@iRow
5027   {
5028     \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5029     \bool_set_true:N \l_@@_final_open_bool
5030   }
5031   {
5032     \cs_if_exist:cTF
5033     {
5034       pgf @ sh @ ns @ \@@_env:
5035       - \int_eval:n { #1 + #3 }
5036       - \int_use:N \l_@@_final_j_int
5037     }
5038     { \int_set:Nn \l_@@_final_i_int { #1 + #3 } }
5039     {
5040       \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5041       \bool_set_true:N \l_@@_final_open_bool
5042     }
5043   }
5044   \group_begin:
5045   \@@_open_shorten:
5046
5047
5048
5049
5050   \int_compare:nNnTF { #2 } = 0
5051   { \color { nicematrix-first-col } }
5052   {
5053     \int_compare:nNnT { #2 } = \g_@@_col_total_int
5054     { \color { nicematrix-last-col } }
5055   }

```



```

5056 \keys_set:nn { NiceMatrix / xdots } { #4 }
5057 \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
5058 \@@_actually_draw_Vdots:
5059 \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).

```

5060 \int_step_inline:nnn { #1 } { #1 + #3 - 1 }
5061 { \cs_set:cpn { @@ _ dotted _ ##1 - #2 } { } }
5062 }

```

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

```

5063 \NewDocumentCommand \@@_rotate: { 0 { } }
5064 {
5065   \peek_remove_spaces:n
5066   {
5067     \bool_gset_true:N \g_@@_rotate_bool
5068     \keys_set:nn { NiceMatrix / rotate } { #1 }
5069   }
5070 }

5071 \keys_define:nn { NiceMatrix / rotate }
5072 {
5073   c .code:n = \bool_gset_true:N \g_@@_rotate_c_bool ,
5074   c .value_forbidden:n = true ,
5075   unknown .code:n = \@@_error:n { Unknown-key-for-rotate }
5076 }

```

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).¹²

```

5077 \cs_new:Npn \@@_double_int_eval:n #1-#2 \q_stop
5078 {
5079   \tl_if_empty:nTF { #2 }
5080   { #1 }
5081   { \@@_double_int_eval_i:n #1-#2 \q_stop }
5082 }
5083 \cs_new:Npn \@@_double_int_eval_i:n #1-#2- \q_stop
5084 { \int_eval:n { #1 } - \int_eval:n { #2 } }

```

¹²Indeed, we want that the user may use the command `\line` in `\CodeAfter` with LaTeX counters in the arguments — with the command `\value`.

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).

```

5085 \hook_gput_code:nnn { begindocument } { . }
5086 {
5087   \tl_set:Nn \l_@@_argspec_tl { 0 { } m m ! 0 { } E { _ ^ : } { { } { } { } } }
5088   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
5089   \exp_args:NNV \NewDocumentCommand \@@_line \l_@@_argspec_tl
5090     {
5091       \group_begin:
5092       \keys_set:nn { NiceMatrix / xdots } { #1 , #4 , down = #5 , up = #6 }
5093       \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
5094       \use:e
5095       {
5096         \@@_line_i:nn
5097         { \@@_double_int_eval:n #2 - \q_stop }
5098         { \@@_double_int_eval:n #3 - \q_stop }
5099       }
5100       \group_end:
5101     }
5102 }
5103 \cs_new_protected:Npn \@@_line_i:nn #1 #2
5104 {
5105   \bool_set_false:N \l_@@_initial_open_bool
5106   \bool_set_false:N \l_@@_final_open_bool
5107   \bool_if:nTF
5108     {
5109       \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #1 }
5110       ||
5111       \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #2 }
5112     }
5113     {
5114       \@@_error:nnn { unknown~cell~for~line~in~CodeAfter } { #1 } { #2 }
5115     }

```

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

```

5116   { \legacy_if:nF { measuring@ } { \@@_draw_line_ii:nn { #1 } { #2 } } }
5117 }
5118 \hook_gput_code:nnn { begindocument } { . }
5119 {
5120   \cs_new_protected:Npx \@@_draw_line_ii:nn #1 #2
5121   {

```

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:.`

```

5122   \c_@@_pgfortikzpicture_tl
5123   \@@_draw_line_iii:nn { #1 } { #2 }
5124   \c_@@_endpgfortikzpicture_tl
5125 }
5126 }

```

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

```

5127 \cs_new_protected:Npn \@@_draw_line_iii:nn #1 #2
5128 {
5129   \pgfrememberpicturepositiononpagetrue
5130   \pgfpointshapeborder { \@@_env: - #1 } { \@@_qpoint:n { #2 } }
5131   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
5132   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
5133   \pgfpointshapeborder { \@@_env: - #2 } { \@@_qpoint:n { #1 } }
5134   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
5135   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y

```

```

5136 \@@_draw_line:
5137 }

```

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`

```

5138 \keys_define:nn { NiceMatrix / RowStyle }
5139 {
5140   cell-space-top-limit .dim_set:N = \l_tmpa_dim ,
5141   cell-space-top-limit .initial:n = \c_zero_dim ,
5142   cell-space-top-limit .value_required:n = true ,
5143   cell-space-bottom-limit .dim_set:N = \l_tmpb_dim ,
5144   cell-space-bottom-limit .initial:n = \c_zero_dim ,
5145   cell-space-bottom-limit .value_required:n = true ,
5146   cell-space-limits .meta:n =
5147   {
5148     cell-space-top-limit = #1 ,
5149     cell-space-bottom-limit = #1 ,
5150   } ,
5151   color .tl_set:N = \l_@@_color_tl ,
5152   color .value_required:n = true ,
5153   bold .bool_set:N = \l_tmpa_bool ,
5154   bold .default:n = true ,
5155   bold .initial:n = false ,
5156   nb-rows .code:n =
5157   \str_if_eq:nnTF { #1 } { * }
5158   { \int_set:Nn \l_@@_key_nb_rows_int { 500 } }
5159   { \int_set:Nn \l_@@_key_nb_rows_int { #1 } } ,
5160   nb-rows .value_required:n = true ,
5161   rowcolor .tl_set:N = \l_tmpa_tl ,
5162   rowcolor .value_required:n = true ,
5163   rowcolor .initial:n = ,
5164   unknown .code:n = \@@_error:n { Unknown~key~for~RowStyle }
5165 }

```

```

5166 \NewDocumentCommand \@@_RowStyle:n { 0 { } m }
5167 {
5168   \group_begin:
5169   \tl_clear:N \l_tmpa_tl % value of \rowcolor
5170   \tl_clear:N \l_@@_color_tl
5171   \int_set:Nn \l_@@_key_nb_rows_int 1
5172   \keys_set:nn { NiceMatrix / RowStyle } { #1 }

```

If the key `rowcolor` has been used.

```

5173   \tl_if_empty:NF \l_tmpa_tl
5174   {

```

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

```

5175     \tl_gput_right:Nx \g_@@_pre_code_before_tl
5176     {

```

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

```

5177     \@@_exp_color_arg:N \@@_rectanglecolor \l_tmpa_tl
5178     { \int_use:N \c@iRow - \int_use:N \c@jCol }
5179     { \int_use:N \c@iRow - * }
5180   }

```

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

```

5181     \int_compare:nNnT \l_@@_key_nb_rows_int > 1

```

```

5182     {
5183         \tl_gput_right:Nx \g_@@_pre_code_before_tl
5184         {
5185             \@@_exp_color_arg:NV \@@_rowcolor \l_tmpa_tl
5186             {
5187                 \int_eval:n { \c@iRow + 1 }
5188                 - \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int - 1 }
5189             }
5190         }
5191     }
5192 }
5193 \tl_gput_right:Nn \g_@@_row_style_tl { \ifnum \c@iRow < }
5194 \tl_gput_right:Nx \g_@@_row_style_tl
5195 { \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int } }
5196 \tl_gput_right:Nn \g_@@_row_style_tl { #2 }
\l_tmpa_dim is the value of the key cell-space-top-limit of \RowStyle.
5197 \dim_compare:nNnT \l_tmpa_dim > \c_zero_dim
5198 {
5199     \tl_gput_right:Nx \g_@@_row_style_tl
5200     {
5201         \tl_gput_right:Nn \exp_not:N \g_@@_cell_after_hook_tl
5202         {
5203             \dim_set:Nn \l_@@_cell_space_top_limit_dim
5204             { \dim_use:N \l_tmpa_dim }
5205         }
5206     }
5207 }
\l_tmpb_dim is the value of the key cell-space-bottom-limit of \RowStyle.
5208 \dim_compare:nNnT \l_tmpb_dim > \c_zero_dim
5209 {
5210     \tl_gput_right:Nx \g_@@_row_style_tl
5211     {
5212         \tl_gput_right:Nn \exp_not:N \g_@@_cell_after_hook_tl
5213         {
5214             \dim_set:Nn \l_@@_cell_space_bottom_limit_dim
5215             { \dim_use:N \l_tmpb_dim }
5216         }
5217     }
5218 }
\l_@@_color_tl is the value of the key color of \RowStyle.
5219 \tl_if_empty:NF \l_@@_color_tl
5220 {
5221     \tl_gput_right:Nx \g_@@_row_style_tl
5222     {
5223         \mode_leave_vertical:
5224         \@@_color:n { \l_@@_color_tl }
5225     }
5226 }
\l_tmpa_bool is the value of the key bold.
5227 \bool_if:NT \l_tmpa_bool
5228 {
5229     \tl_gput_right:Nn \g_@@_row_style_tl
5230     {
5231         \if_mode_math:
5232             \c_math_toggle_token
5233             \bfseries \boldmath
5234             \c_math_toggle_token
5235         \else:
5236             \bfseries \boldmath
5237         \fi:
5238     }
5239 }

```

```

5240 \tl_gput_right:Nn \g_@@_row_style_tl { \fi }
5241 \group_end:
5242 \g_@@_row_style_tl
5243 \ignorespaces
5244 }

```

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).

```

5245 \cs_new_protected:Npn \@@_add_to_colors_seq:nn #1 #2
5246 {

```

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.

```

5247 \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`.

```

5248 \str_if_in:nnF { #1 } { !! }
5249 {
5250 \seq_map_indexed_inline:Nn \g_@@_colors_seq
5251 { \tl_if_eq:nnT { #1 } { ##2 } { \int_set:Nn \l_tmpa_int { ##1 } } }
5252 }
5253 \int_compare:nNnTF \l_tmpa_int = \c_zero_int

```

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

```

5254 {
5255 \seq_gput_right:Nn \g_@@_colors_seq { #1 }
5256 \tl_gset:cx { g_@@_color _ \seq_count:N \g_@@_colors_seq _ tl } { #2 }
5257 }

```

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

```

5258 { \tl_gput_right:cx { g_@@_color _ \int_use:N \l_tmpa_int _tl } { #2 } }
5259 }

5260 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { x n }
5261 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { x x }

```

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

```

5262 \cs_new_protected:Npn \@@_clip_with_rounded_corners:
5263 {
5264 \dim_compare:nNnT \l_@@_tab_rounded_corners_dim > \c_zero_dim
5265 {

```

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

```

5266     \group_begin:
5267     \pgfsetcornersarced
5268     {
5269         \pgfpoint
5270         { \l_@@_tab_rounded_corners_dim }
5271         { \l_@@_tab_rounded_corners_dim }
5272     }

```

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.

```

5273     \bool_if:NTF \l_@@_hvlines_bool
5274     {
5275         \pgfpathrectanglecorners
5276         {
5277             \pgfpointadd
5278             { \@@_qpoint:n { row-1 } }
5279             { \pgfpoint { 0.5 \arrayrulewidth } { \c_zero_dim } }
5280         }
5281         {
5282             \pgfpointadd
5283             {
5284                 \@@_qpoint:n
5285                 { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
5286             }
5287             { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
5288         }
5289     }
5290     {
5291         \pgfpathrectanglecorners
5292         { \@@_qpoint:n { row-1 } }
5293         {
5294             \pgfpointadd
5295             {
5296                 \@@_qpoint:n
5297                 { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
5298             }
5299             { \pgfpoint \c_zero_dim \arrayrulewidth }
5300         }
5301     }
5302     \pgfusepath { clip }
5303     \group_end:

```

The TeX group was for `\pgfsetcornersarced`.

```

5304     }
5305 }

```

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`).

```

5306 \cs_new_protected:Npn \@@_actually_color:
5307 {
5308     \pgfpicture
5309     \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.

```

5310     \@@_clip_with_rounded_corners:
5311     \seq_map_indexed_inline:Nn \g_@@_colors_seq
5312     {
5313         \begin { pgfscope }

```

```

5314         \@@_color_opacity ##2
5315         \use:c { g_@@_color _ ##1 _t1 }
5316         \tl_gclear:c { g_@@_color _ ##1 _t1 }
5317         \pgfusepath { fill }
5318     \end { pgfscope }
5319 }
5320 \endpgfpicture
5321 }

```

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

```

5322 \cs_new_protected:Npn \@@_color_opacity
5323 {
5324     \peek_meaning:NTF [
5325         { \@@_color_opacity:w }
5326         { \@@_color_opacity:w [ ] }
5327     }

```

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.

```

5328 \cs_new_protected:Npn \@@_color_opacity:w [ #1 ]
5329 {
5330     \tl_clear:N \l_tmpa_tl
5331     \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.

```

5332     \tl_if_empty:NF \l_tmpa_tl { \exp_args:NV \pgfsetfillopacity \l_tmpa_tl }
5333     \tl_if_empty:NTF \l_tmpb_tl
5334         { \@declaredcolor }
5335         { \use:x { \exp_not:N \@undeclaredcolor [ \l_tmpb_tl ] } }
5336 }

```

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

```

5337 \keys_define:nn { nicematrix / color-opacity }
5338 {
5339     opacity .tl_set:N          = \l_tmpa_tl ,
5340     opacity .value_required:n = true
5341 }

5342 \cs_new_protected:Npn \@@_cartesian_color:nn #1 #2
5343 {
5344     \tl_set:Nn \l_@@_rows_tl { #1 }
5345     \tl_set:Nn \l_@@_cols_tl { #2 }
5346     \@@_cartesian_path:
5347 }

```

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

```

5348 \NewDocumentCommand \@@_rowcolor { 0 { } m m }
5349 {
5350     \tl_if_blank:nF { #2 }
5351     {
5352         \@@_add_to_colors_seq:xn
5353         { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5354         { \@@_cartesian_color:nn { #3 } { - } }
5355     }
5356 }

```

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

```

5357 \NewDocumentCommand \@@_columncolor { 0 { } m m }
5358 {
5359   \tl_if_blank:nF { #2 }
5360   {
5361     \@@_add_to_colors_seq:xn
5362     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5363     { \@@_cartesian_color:nn { - } { #3 } }
5364   }
5365 }

```

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

```

5366 \NewDocumentCommand \@@_rectanglecolor { 0 { } m m m }
5367 {
5368   \tl_if_blank:nF { #2 }
5369   {
5370     \@@_add_to_colors_seq:xn
5371     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5372     { \@@_rectanglecolor:nnn { #3 } { #4 } { 0 pt } }
5373   }
5374 }

```

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

```

5375 \NewDocumentCommand \@@_roundedrectanglecolor { 0 { } m m m m }
5376 {
5377   \tl_if_blank:nF { #2 }
5378   {
5379     \@@_add_to_colors_seq:xn
5380     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5381     { \@@_rectanglecolor:nnn { #3 } { #4 } { #5 } }
5382   }
5383 }

```

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

```

5384 \cs_new_protected:Npn \@@_rectanglecolor:nnn #1 #2 #3
5385 {
5386   \@@_cut_on_hyphen:w #1 \q_stop
5387   \tl_clear_new:N \l_@@_tmpc_tl
5388   \tl_clear_new:N \l_@@_tmpd_tl
5389   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
5390   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
5391   \@@_cut_on_hyphen:w #2 \q_stop
5392   \tl_set:Nx \l_@@_rows_tl { \l_@@_tmpc_tl - \l_tmpa_tl }
5393   \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.

```

5394   \@@_cartesian_path:n { #3 }
5395 }

```

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

```

5396 \NewDocumentCommand \@@_cellcolor { 0 { } m m }
5397 {
5398   \clist_map_inline:nn { #3 }
5399   { \@@_rectanglecolor [ #1 ] { #2 } { ##1 } { ##1 } }
5400 }

```

```

5401 \NewDocumentCommand \@@_chessboardcolors { 0 { } m m }
5402 {
5403   \int_step_inline:nn { \int_use:N \c@iRow }

```



```

5404 {
5405   \int_step_inline:nn { \int_use:N \c@jCol }
5406   {
5407     \int_if_even:nTF { ####1 + #1 }
5408     { \@@_cellcolor [ #1 ] { #2 } }
5409     { \@@_cellcolor [ #1 ] { #3 } }
5410     { ##1 - ####1 }
5411   }
5412 }
5413 }

```

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”.

```

5414 \NewDocumentCommand \@@_arraycolor { 0 { } m }
5415 {
5416   \@@_rectanglecolor [ #1 ] { #2 }
5417   { 1 - 1 }
5418   { \int_use:N \c@iRow - \int_use:N \c@jCol }
5419 }

```

```

5420 \keys_define:nn { NiceMatrix / rowcolors }
5421 {
5422   respect-blocks .bool_set:N = \l_@@_respect_blocks_bool ,
5423   respect-blocks .default:n = true ,
5424   cols .tl_set:N = \l_@@_cols_tl ,
5425   restart .bool_set:N = \l_@@_rowcolors_restart_bool ,
5426   restart .default:n = true ,
5427   unknown .code:n = \@@_error:n { Unknown~key~for~rowcolors }
5428 }

```

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*.

```

5429 \NewDocumentCommand \@@_rowlistcolors { 0 { } m m 0 { } }
5430 {

```

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

```

5431   \group_begin:
5432   \seq_clear_new:N \l_@@_colors_seq
5433   \seq_set_split:Nnn \l_@@_colors_seq { , } { #3 }
5434   \tl_clear_new:N \l_@@_cols_tl
5435   \tl_set:Nn \l_@@_cols_tl { - }
5436   \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).

```

5437   \int_zero_new:N \l_@@_color_int
5438   \int_set:Nn \l_@@_color_int 1
5439   \bool_if:NT \l_@@_respect_blocks_bool
5440   {

```

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`).

```

5441   \seq_set_eq:NN \l_tmpb_seq \g_@@_pos_of_blocks_seq
5442   \seq_set_filter:Nnn \l_tmpa_seq \l_tmpb_seq
5443   { \@@_not_in_exterior_p:nnnnn #1 }
5444 }

```

```

5445 \pgfpicture
5446 \pgf@relevantforpicturesizefalse

```

#2 is the list of intervals of rows.

```

5447 \clist_map_inline:nn { #2 }
5448 {
5449   \tl_set:Nn \l_tmpa_tl { ##1 }
5450   \tl_if_in:NnTF \l_tmpa_tl { - }
5451   { \@@_cut_on_hyphen:w ##1 \q_stop }
5452   { \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.

```

5453   \int_set:Nn \l_tmpa_int \l_tmpa_tl
5454   \int_set:Nn \l_@@_color_int
5455   { \bool_if:NTF \l_@@_rowcolors_restart_bool 1 \l_tmpa_tl }
5456   \int_zero_new:N \l_@@_tmpc_int
5457   \int_set:Nn \l_@@_tmpc_int \l_tmpb_tl
5458   \int_do_until:nNnn \l_tmpa_int > \l_@@_tmpc_int
5459   {

```

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

```

5460     \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).

```

5461     \bool_if:NT \l_@@_respect_blocks_bool
5462     {
5463       \seq_set_filter:Nn \l_tmpb_seq \l_tmpa_seq
5464       { \@@_intersect_our_row_p:nnnnn ###1 }
5465       \seq_map_inline:Nn \l_tmpb_seq { \@@_rowcolors_i:nnnnn ###1 }

```

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

```

5466     }
5467     \tl_set:Nx \l_@@_rows_tl
5468     { \int_use:N \l_tmpa_int - \int_use:N \l_tmpb_int }

```

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

```

5469     \tl_clear_new:N \l_@@_color_tl
5470     \tl_set:Nx \l_@@_color_tl
5471     {
5472       \@@_color_index:n
5473       {
5474         \int_mod:nn
5475         { \l_@@_color_int - 1 }
5476         { \seq_count:N \l_@@_colors_seq }
5477         + 1
5478       }
5479     }
5480     \tl_if_empty:NF \l_@@_color_tl
5481     {
5482       \@@_add_to_colors_seq:xx
5483       { \tl_if_blank:nF { #1 } { [ #1 ] } { \l_@@_color_tl } }
5484       { \@@_cartesian_color:nn { \l_@@_rows_tl } { \l_@@_cols_tl } }
5485     }
5486     \int_incr:N \l_@@_color_int
5487     \int_set:Nn \l_tmpa_int { \l_tmpb_int + 1 }
5488   }
5489 }
5490 \endpgfpicture
5491 \group_end:
5492 }

```

The command `\@@_color_index:n` peaks 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.

```

5493 \cs_new:Npn \@@_color_index:n #1
5494 {

```

```

5495 \str_if_eq:eeTF { \seq_item:Nn \l_@@_colors_seq { #1 } } { = }
5496 { \@@_color_index:n { #1 - 1 } }
5497 { \seq_item:Nn \l_@@_colors_seq { #1 } }
5498 }

```

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 currying.

```

5499 \NewDocumentCommand \@@_rowcolors { 0 { } m m m }
5500 { \@@_rowlistcolors [ #1 ] { #2 } { { #3 } , { #4 } } }

5501 \cs_new_protected:Npn \@@_rowcolors_i:nnnnn #1 #2 #3 #4 #5
5502 {
5503   \int_compare:nNt { #3 } > \l_tmpb_int
5504   { \int_set:Nn \l_tmpb_int { #3 } }
5505 }

5506 \prg_new_conditional:Nnn \@@_not_in_exterior:nnnnn p
5507 {
5508   \bool_lazy_or:nnTF
5509   { \int_compare_p:nNn { #4 } = \c_zero_int }
5510   { \int_compare_p:nNn { #2 } = { \int_eval:n { \c@jCol + 1 } } }
5511   \prg_return_false:
5512   \prg_return_true:
5513 }

```

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

```

5514 \prg_new_conditional:Nnn \@@_intersect_our_row:nnnnn p
5515 {
5516   \bool_if:nTF
5517   {
5518     \int_compare_p:n { #1 <= \l_tmpa_int }
5519     &&
5520     \int_compare_p:n { \l_tmpa_int <= #3 }
5521   }
5522   \prg_return_true:
5523   \prg_return_false:
5524 }

```

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`).

```

5525 \cs_new_protected:Npn \@@_cartesian_path:n #1
5526 {
5527   \bool_lazy_and:nnT
5528   { ! \seq_if_empty_p:N \l_@@_corners_cells_seq }
5529   { \dim_compare_p:nNn { #1 } = \c_zero_dim }
5530   {
5531     \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
5532     \@@_expand_clist:NN \l_@@_rows_tl \c@iRow
5533   }

```

We begin the loop over the columns.

```

5534 \clist_map_inline:Nn \l_@@_cols_tl
5535 {
5536   \tl_set:Nn \l_tmpa_tl { ##1 }
5537   \tl_if_in:NnTF \l_tmpa_tl { - }

```

```

5538     { \@@_cut_on_hyphen:w ##1 \q_stop }
5539     { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
5540 \bool_lazy_or:nnT
5541     { \tl_if_blank_p:V \l_tmpa_tl }
5542     { \str_if_eq_p:Vn \l_tmpa_tl { * } }
5543     { \tl_set:Nn \l_tmpa_tl { 1 } }
5544 \bool_lazy_or:nnT
5545     { \tl_if_blank_p:V \l_tmpb_tl }
5546     { \str_if_eq_p:Vn \l_tmpb_tl { * } }
5547     { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@jCol } }
5548 \int_compare:nNnT \l_tmpb_tl > \c@jCol
5549     { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@jCol } }

```

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

```

5550     \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.

```

5551     \@@_qpoint:n { col - \l_tmpa_tl }
5552     \int_compare:nNnTF \l_@@_first_col_int = \l_tmpa_tl
5553     { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
5554     { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
5555     \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
5556     \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }

```

We begin the loop over the rows.

```

5557     \clist_map_inline:Nn \l_@@_rows_tl
5558     {
5559         \tl_set:Nn \l_tmpa_tl { #####1 }
5560         \tl_if_in:NnTF \l_tmpa_tl { - }
5561         { \@@_cut_on_hyphen:w #####1 \q_stop }
5562         { \@@_cut_on_hyphen:w #####1 - #####1 \q_stop }
5563         \tl_if_empty:NT \l_tmpa_tl { \tl_set:Nn \l_tmpa_tl { 1 } }
5564         \tl_if_empty:NT \l_tmpb_tl
5565         { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@iRow } }
5566         \int_compare:nNnT \l_tmpb_tl > \c@iRow
5567         { \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`.

```

5568         \seq_if_in:NxF \l_@@_corners_cells_seq
5569         { \l_tmpa_tl - \l_@@_tmpc_tl }
5570         {
5571             \@@_qpoint:n { row - \int_eval:n { \l_tmpb_tl + 1 } }
5572             \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
5573             \@@_qpoint:n { row - \l_tmpa_tl }
5574             \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
5575             \pgfsetcornersarced { \pgfpoint { #1 } { #1 } }
5576             \pgfpathrectanglecorners
5577             { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
5578             { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5579         }
5580     }
5581 }
5582 }

```

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`).

```

5583 \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 `\l_@@_cols_tl` will be replaced by 2,4,5,6,8,9,10.

```

5584 \cs_new_protected:Npn \@@_expand_clist:NN #1 #2
5585 {
5586   \clist_set_eq:NN \l_tmpa_clist #1
5587   \clist_clear:N #1
5588   \clist_map_inline:Nn \l_tmpa_clist
5589   {
5590     \tl_set:Nn \l_tmpa_tl { ##1 }
5591     \tl_if_in:NnTF \l_tmpa_tl { - }
5592     { \@@_cut_on_hyphen:w ##1 \q_stop }
5593     { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
5594     \bool_lazy_or:nnT
5595     { \tl_if_blank_p:V \l_tmpa_tl }
5596     { \str_if_eq_p:Vn \l_tmpa_tl { * } }
5597     { \tl_set:Nn \l_tmpa_tl { 1 } }
5598     \bool_lazy_or:nnT
5599     { \tl_if_blank_p:V \l_tmpb_tl }
5600     { \str_if_eq_p:Vn \l_tmpb_tl { * } }
5601     { \tl_set:Nx \l_tmpb_tl { \int_use:N #2 } }
5602     \int_compare:nNnT \l_tmpb_tl > #2
5603     { \tl_set:Nx \l_tmpb_tl { \int_use:N #2 } }
5604     \int_step_inline:nnn \l_tmpa_tl \l_tmpb_tl
5605     { \clist_put_right:Nn #1 { #####1 } }
5606   }
5607 }

```

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

```

5608 \NewDocumentCommand \@@_cellcolor_tabular { 0 { } m }
5609 {
5610   \tl_gput_right:Nx \g_@@_pre_code_before_tl
5611   {

```

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`).

```

5612     \@@_cellcolor [ #1 ] { \exp_not:n { #2 } }
5613     { \int_use:N \c@iRow - \int_use:N \c@jCol }
5614   }
5615   \ignorespaces
5616 }

```

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

```

5617 \NewDocumentCommand \@@_rowcolor_tabular { 0 { } m }
5618 {
5619   \tl_gput_right:Nx \g_@@_pre_code_before_tl
5620   {
5621     \@@_rectanglecolor [ #1 ] { \exp_not:n { #2 } }
5622     { \int_use:N \c@iRow - \int_use:N \c@jCol }
5623     { \int_use:N \c@iRow - \exp_not:n { \int_use:N \c@jCol } }
5624   }
5625   \ignorespaces
5626 }

```

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).

```

5627 \NewDocumentCommand { \@@_rowcolors_tabular } { 0 { } m m }
5628 { \@@_rowlistcolors_tabular [ #1 ] { #2 , #3 } }

```

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

```

5629 \NewDocumentCommand { \@@_rowlistcolors_tabular } { 0 { } m 0 { } }

```

```

5630 {
5631   \peek_remove_spaces:n
5632   {
5633     \tl_gput_right:Nx \g_@@_pre_code_before_tl
5634     {
5635       \@@_rowlistcolors
5636       [ #1 ] { \int_use:N \c@iRow } { #2 }
5637       [ restart, cols = \int_use:N \c@jCol - , #3 ]
5638     }
5639   }
5640 }

```

```

5641 \NewDocumentCommand \@@_columncolor_preamble { 0 { } m }
5642 {

```

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).

```

5643   \int_compare:nNnT \c@jCol > \g_@@_col_total_int
5644   {

```

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).

```

5645     \tl_gput_left:Nx \g_@@_pre_code_before_tl
5646     {
5647       \exp_not:N \columncolor [ #1 ]
5648       { \exp_not:n { #2 } } { \int_use:N \c@jCol }
5649     }
5650   }
5651 }

```

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`.

```

5652 \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`.

```

5653 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix:n #1
5654 {
5655   \int_compare:nNnTF \l_@@_first_col_int = 0
5656   { \@@_OnlyMainNiceMatrix_i:n { #1 } }
5657   {
5658     \int_compare:nNnTF \c@jCol = 0
5659     {
5660       \int_compare:nNnF \c@iRow = { -1 }
5661       { \int_compare:nNnF \c@iRow = { \l_@@_last_row_int - 1 } { #1 } }
5662     }

```

```

5663     { \@@_OnlyMainNiceMatrix_i:n { #1 } }
5664   }
5665 }

```

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.

```

5666 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix_i:n #1
5667 {
5668   \int_compare:nNnF \c@iRow = 0
5669   {
5670     \int_compare:nNnF \c@iRow = \l_@@_last_row_int
5671     {
5672       \int_compare:nNnT \c@jCol > \c_zero_int
5673       { \bool_if:NF \l_@@_in_last_col_bool { #1 } }
5674     }
5675   }
5676 }

```

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.

```

5677 \keys_define:nn { NiceMatrix / Rules }
5678 {
5679   position .int_set:N = \l_@@_position_int ,
5680   position .value_required:n = true ,
5681   start .int_set:N = \l_@@_start_int ,
5682   start .initial:n = 1 ,
5683   end .code:n =
5684     \bool_lazy_or:nnTF
5685     { \tl_if_empty_p:n { #1 } }
5686     { \str_if_eq_p:nn { #1 } { last } }
5687     { \int_set_eq:NN \l_@@_end_int \c@jCol }
5688     { \int_set:Nn \l_@@_end_int { #1 } }
5689 }

```

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.

```

5690 \keys_define:nn { NiceMatrix / RulesBis }
5691 {
5692   multiplicity .int_set:N = \l_@@_multiplicity_int ,
5693   multiplicity .initial:n = 1 ,
5694   dotted .bool_set:N = \l_@@_dotted_bool ,
5695   dotted .initial:n = false ,
5696   dotted .default:n = true ,

```

We want that, even when the rule has been defined with TikZ by the key `tikz`, the user has still the possibility to change the color of the rule with the key `color` (in the command `\Hline`, not in the key `tikz` of the command `\Hline`). The main use is, when the user has defined its own command `\MyDashedLine` by `\newcommand{\MyDashedRule}{\Hline[tikz=dashed]}`, to give the ability to write `\MyDashedRule[color=red]`.

```

5697 color .code:n =
5698   \@@_set_CT@arc@:n { #1 }
5699   \tl_set:Nn \l_@@_rule_color_tl { #1 } ,
5700 color .value_required:n = true ,
5701 sep-color .code:n = \@@_set_CT@drsc@:n { #1 } ,
5702 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.

```

5703 tikz .code:n =
5704   \IfPackageLoadedTF { tikz }
5705     { \clist_put_right:Nn \l_@@_tikz_rule_tl { #1 } }
5706     { \@@_error:n { tikz-without-tikz } } ,
5707 tikz .value_required:n = true ,
5708 total-width .dim_set:N = \l_@@_rule_width_dim ,
5709 total-width .value_required:n = true ,
5710 width .meta:n = { total-width = #1 } ,
5711 unknown .code:n = \@@_error:n { Unknow-key-for-RulesBis }
5712 }

```

The vertical rules

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

```

5713 \cs_new_protected:Npn \@@_vline:n #1
5714 {

```

The group is for the options.

```

5715   \group_begin:
5716   \int_zero_new:N \l_@@_end_int
5717   \int_set_eq:NN \l_@@_end_int \c_iRow
5718   \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).

```

5719   \int_compare:nNnT \l_@@_position_int < { \c_jCol + 2 }
5720     \@@_vline_i:
5721   \group_end:
5722 }

```

```

5723 \cs_new_protected:Npn \@@_vline_i:
5724 {
5725   \int_zero_new:N \l_@@_local_start_int
5726   \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`.

```

5727   \tl_set:Nx \l_tmpb_tl { \int_eval:n \l_@@_position_int }
5728   \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
5729     \l_tmpa_tl
5730   {

```

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.

```

5731     \bool_gset_true:N \g_tmpa_bool
5732     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
5733       { \@@_test_vline_in_block:nnnnn ##1 }
5734     \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
5735       { \@@_test_vline_in_block:nnnnn ##1 }
5736     \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
5737       { \@@_test_vline_in_stroken_block:nnnn ##1 }
5738     \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_v:
5739     \bool_if:NTF \g_tmpa_bool

```



```

5740     {
5741         \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.

```

5742         { \int_set:Nn \l_@@_local_start_int \l_tmpa_tl }
5743     }
5744     {
5745         \int_compare:nNnT \l_@@_local_start_int > 0
5746         {
5747             \int_set:Nn \l_@@_local_end_int { \l_tmpa_tl - 1 }
5748             \@@_vline_ii:
5749             \int_zero:N \l_@@_local_start_int
5750         }
5751     }
5752 }
5753 \int_compare:nNnT \l_@@_local_start_int > 0
5754 {
5755     \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
5756     \@@_vline_ii:
5757 }
5758 }

5759 \cs_new_protected:Npn \@@_test_in_corner_v:
5760 {
5761     \int_compare:nNnTF \l_tmpb_tl = { \int_eval:n { \c@jCol + 1 } }
5762     {
5763         \seq_if_in:NxT
5764         \l_@@_corners_cells_seq
5765         { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
5766         { \bool_set_false:N \g_tmpa_bool }
5767     }
5768     {
5769         \seq_if_in:NxT
5770         \l_@@_corners_cells_seq
5771         { \l_tmpa_tl - \l_tmpb_tl }
5772         {
5773             \int_compare:nNnTF \l_tmpb_tl = 1
5774             { \bool_set_false:N \g_tmpa_bool }
5775             {
5776                 \seq_if_in:NxT
5777                 \l_@@_corners_cells_seq
5778                 { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
5779                 { \bool_set_false:N \g_tmpa_bool }
5780             }
5781         }
5782     }
5783 }

5784 \cs_new_protected:Npn \@@_vline_ii:
5785 {
5786     \tl_clear:N \l_@@_tikz_rule_tl
5787     \keys_set:nV { NiceMatrix / RulesBis } \l_@@_other_keys_tl
5788     \bool_if:NTF \l_@@_dotted_bool
5789     \@@_vline_iv:
5790     {
5791         \tl_if_empty:NTF \l_@@_tikz_rule_tl
5792         \@@_vline_iii:
5793         \@@_vline_v:
5794     }
5795 }

```

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

```

5796 \cs_new_protected:Npn \@@_vline_iii:
5797 {
5798   \pgfpicture
5799   \pgfrememberpicturepositiononpagetrue
5800   \pgf@relevantforpicturesizefalse
5801   \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
5802   \dim_set_eq:NN \l_tmpa_dim \pgf@y
5803   \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
5804   \dim_set:Nn \l_tmpb_dim
5805     {
5806       \pgf@x
5807       - 0.5 \l_@@_rule_width_dim
5808       +
5809       ( \arrayrulewidth * \l_@@_multiplicity_int
5810         + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
5811     }
5812   \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
5813   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
5814   \bool_lazy_all:nT
5815     {
5816       { \int_compare_p:nNn \l_@@_multiplicity_int > 1 }
5817       { \cs_if_exist_p:N \CT@drsc@ }
5818       { ! \tl_if_blank_p:V \CT@drsc@ }
5819     }
5820     {
5821       \group_begin:
5822       \CT@drsc@
5823       \dim_add:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }
5824       \dim_sub:Nn \l_@@_tmpc_dim { 0.5 \arrayrulewidth }
5825       \dim_set:Nn \l_@@_tmpd_dim
5826         {
5827           \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
5828           * ( \l_@@_multiplicity_int - 1 )
5829         }
5830       \pgfpathrectanglecorners
5831         { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
5832         { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
5833       \pgfusepath { fill }
5834       \group_end:
5835     }
5836   \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
5837   \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
5838   \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
5839     {
5840       \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
5841       \dim_sub:Nn \l_tmpb_dim \doublerulesep
5842       \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
5843       \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
5844     }
5845   \CT@arc@
5846   \pgfsetlinewidth { 1.1 \arrayrulewidth }
5847   \pgfsetrectcap
5848   \pgfusepathqstroke
5849   \endpgfpicture
5850 }

```

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

```

5851 \cs_new_protected:Npn \@@_vline_iv:
5852 {
5853   \pgfpicture
5854   \pgfrememberpicturepositiononpagetrue
5855   \pgf@relevantforpicturesizefalse

```

```

5856 \l_@@_qpoint:n { col - \int_use:N \l_@@_position_int }
5857 \dim_set:Nn \l_@@_x_initial_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
5858 \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
5859 \l_@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
5860 \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
5861 \l_@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
5862 \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
5863 \CT@arc@
5864 \l_@@_draw_line:
5865 \endpgfpicture
5866 }

```

The following code is for the case when the user uses the key `tikz`.

```

5867 \cs_new_protected:Npn \l_@@_vline_v:
5868 {
5869   \begin{tikzpicture}
5870     % added 2023/09/25

```

By default, the color defined by `\arrayrulecolor` or by `rules/color` will be used, but it's still possible to change the color by using the key `color` or, of course, the key `color` inside the key `tikz` (that is to say the key `color` provided by PGF).

```

5871 \CT@arc@
5872 \tl_if_empty:NF \l_@@_rule_color_tl
5873 { \tl_put_right:Nx \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
5874 \pgfrememberpicturepositiononpagetrue
5875 \pgf@relevantforpicturesizefalse
5876 \l_@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
5877 \dim_set_eq:NN \l_tmpa_dim \pgf@y
5878 \l_@@_qpoint:n { col - \int_use:N \l_@@_position_int }
5879 \dim_set:Nn \l_tmpb_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
5880 \l_@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
5881 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
5882 \exp_args:NV \tikzset \l_@@_tikz_rule_tl
5883 \use:x { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
5884 ( \l_tmpb_dim , \l_tmpa_dim ) --
5885 ( \l_tmpb_dim , \l_@@_tmpc_dim ) ;
5886 \end{tikzpicture}
5887 }

```

The command `\l_@@_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).

```

5888 \cs_new_protected:Npn \l_@@_draw_vlines:
5889 {
5890   \int_step_inline:nnn
5891   {
5892     \bool_if:nTF { ! \g_@@_delims_bool && ! \l_@@_except_borders_bool }
5893     1 2
5894   }
5895   {
5896     \bool_if:nTF { ! \g_@@_delims_bool && ! \l_@@_except_borders_bool }
5897     { \int_eval:n { \c@jCol + 1 } }
5898     \c@jCol
5899   }
5900   {
5901     \tl_if_eq:NnF \l_@@_vlines_clist { all }
5902     { \clist_if_in:NnT \l_@@_vlines_clist { ##1 } }
5903     { \l_@@_vline:n { position = ##1 , total-width = \arrayrulewidth } }
5904   }
5905 }

```

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}`.

```
5906 \cs_new_protected:Npn \@@_hline:n #1
5907 {
```

The group is for the options.

```
5908   \group_begin:
5909   \int_zero_new:N \l_@@_end_int
5910   \int_set_eq:NN \l_@@_end_int \c@jCol
5911   \keys_set_known:nN { NiceMatrix / Rules } { #1 } \l_@@_other_keys_tl
5912   \@@_hline_i:
5913   \group_end:
5914 }
```

```
5915 \cs_new_protected:Npn \@@_hline_i:
5916 {
5917   \int_zero_new:N \l_@@_local_start_int
5918   \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`.

```
5919   \tl_set:Nx \l_tmpa_tl { \int_use:N \l_@@_position_int }
5920   \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
5921     \l_tmpb_tl
5922   {
```

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.

```
5923     \bool_gset_true:N \g_tmpa_bool
5924     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
5925       { \@@_test_hline_in_block:nnnnn ##1 }
5926     \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
5927       { \@@_test_hline_in_block:nnnnn ##1 }
5928     \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
5929       { \@@_test_hline_in_stroken_block:nnnn ##1 }
5930     \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_h:
5931     \bool_if:NTF \g_tmpa_bool
5932       {
5933         \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.

```
5934         { \int_set:Nn \l_@@_local_start_int \l_tmpb_tl }
5935       }
5936     {
5937       \int_compare:nNnT \l_@@_local_start_int > 0
5938       {
5939         \int_set:Nn \l_@@_local_end_int { \l_tmpb_tl - 1 }
5940         \@@_hline_ii:
5941         \int_zero:N \l_@@_local_start_int
5942       }
5943     }
5944   }
5945   \int_compare:nNnT \l_@@_local_start_int > 0
5946   {
5947     \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
5948     \@@_hline_ii:
5949   }
5950 }
```

```

5951 \cs_new_protected:Npn \@@_test_in_corner_h:
5952 {
5953   \int_compare:nNnTF \l_tmpa_tl = { \int_eval:n { \c@iRow + 1 } }
5954   {
5955     \seq_if_in:NxT
5956     \l_@@_corners_cells_seq
5957     { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
5958     { \bool_set_false:N \g_tmpa_bool }
5959   }
5960   {
5961     \seq_if_in:NxT
5962     \l_@@_corners_cells_seq
5963     { \l_tmpa_tl - \l_tmpb_tl }
5964     {
5965       \int_compare:nNnTF \l_tmpa_tl = 1
5966       { \bool_set_false:N \g_tmpa_bool }
5967       {
5968         \seq_if_in:NxT
5969         \l_@@_corners_cells_seq
5970         { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
5971         { \bool_set_false:N \g_tmpa_bool }
5972       }
5973     }
5974   }
5975 }

```

```

5976 \cs_new_protected:Npn \@@_hline_ii:
5977 {
5978   \tl_clear:N \l_@@_tikz_rule_tl
5979   \keys_set:nV { NiceMatrix / RulesBis } \l_@@_other_keys_tl
5980   \bool_if:NTF \l_@@_dotted_bool
5981   \@@_hline_iv:
5982   {
5983     \tl_if_empty:NTF \l_@@_tikz_rule_tl
5984     \@@_hline_iii:
5985     \@@_hline_v:
5986   }
5987 }

```

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

```

5988 \cs_new_protected:Npn \@@_hline_iii:
5989 {
5990   \pgfpicture
5991   \pgfrememberpicturepositiononpagetrue
5992   \pgf@relevantforpicturesizefalse
5993   \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
5994   \dim_set_eq:NN \l_tmpa_dim \pgf@x
5995   \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
5996   \dim_set:Nn \l_tmpb_dim
5997   {
5998     \pgf@y
5999     - 0.5 \l_@@_rule_width_dim
6000     +
6001     ( \arrayrulewidth * \l_@@_multiplicity_int
6002       + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6003   }
6004   \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6005   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
6006   \bool_lazy_all:nT
6007   {
6008     { \int_compare_p:nNn \l_@@_multiplicity_int > 1 }
6009     { \cs_if_exist_p:N \CT@drsc@ }

```

```

6010     { ! \tl_if_blank_p:V \CT@drsc@ }
6011   }
6012   {
6013     \group_begin:
6014     \CT@drsc@
6015     \dim_set:Nn \l_@@_tmpd_dim
6016     {
6017       \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
6018       * ( \l_@@_multiplicity_int - 1 )
6019     }
6020     \pgfpathrectanglecorners
6021     { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6022     { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
6023     \pgfusepathqfill
6024     \group_end:
6025   }
6026   \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6027   \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
6028   \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6029   {
6030     \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
6031     \dim_sub:Nn \l_tmpb_dim \doublerulesep
6032     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6033     \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
6034   }
6035   \CT@arc@
6036   \pgfsetlinewidth { 1.1 \arrayrulewidth }
6037   \pgfsetrectcap
6038   \pgfusepathqstroke
6039   \endpgfpicture
6040 }

```

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}
```

$$\left[\begin{array}{cccc} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ \hdots & \hdots & \hdots & \hdots \\ 1 & 2 & 3 & 4 \end{array} \right]$$

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}
```

$$\left[\begin{array}{cccc} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ \hdots & \hdots & \hdots & \hdots \\ 1 & 2 & 3 & 4 \end{array} \right]$$

```
6041 \cs_new_protected:Npn \l_@@_hline_iv:
```

```
6042 {
```

```
6043   \pgfpicture
```

```
6044   \pgfrememberpicturepositiononpagetrue
```

```
6045   \pgf@relevantforpicturesizefalse
```

```
6046   \l_@@_qpoint:n { row - \int_use:N \l_@@_position_int }
```

```
6047   \dim_set:Nn \l_@@_y_initial_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
```

```
6048   \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
```

```
6049   \l_@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
```

```
6050   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
```

```
6051   \int_compare:nNnT \l_@@_local_start_int = 1
```

```
6052   {
```

```

6053 \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
6054 \bool_if:NF \g_@@_delims_bool
6055 { \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.

```

6056 \tl_if_eq:NnF \g_@@_left_delim_tl (
6057 { \dim_add:Nn \l_@@_x_initial_dim { 0.5 \l_@@_xdots_inter_dim } }
6058 }
6059 \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6060 \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
6061 \int_compare:nNnT \l_@@_local_end_int = \c@jCol
6062 {
6063 \dim_add:Nn \l_@@_x_final_dim \l_@@_right_margin_dim
6064 \bool_if:NF \g_@@_delims_bool
6065 { \dim_add:Nn \l_@@_x_final_dim \arraycolsep }
6066 \tl_if_eq:NnF \g_@@_right_delim_tl )
6067 { \dim_gsub:Nn \l_@@_x_final_dim { 0.5 \l_@@_xdots_inter_dim } }
6068 }
6069 \CT@arc@
6070 \@@_draw_line:
6071 \endpgfpicture
6072 }

```

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`).

```

6073 \cs_new_protected:Npn \@@_hline_v:
6074 {
6075 \begin { tikzpicture }
6076 % added 2023/09/25

```

By default, the color defined by `\arrayrulecolor` or by `rules/color` will be used, but it's still possible to change the color by using the key `color` or, of course, the key `color` inside the key `tikz` (that is to say the key `color` provided by PGF).

```

6077 \CT@arc@
6078 \tl_if_empty:NF \l_@@_rule_color_tl
6079 { \tl_put_right:Nx \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6080 \pgfrememberpicturepositiononpagetrue
6081 \pgf@relevantforpicturesizefalse
6082 \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6083 \dim_set_eq:NN \l_tmpa_dim \pgf@x
6084 \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6085 \dim_set:Nn \l_tmpb_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
6086 \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6087 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
6088 \exp_args:NV \tikzset \l_@@_tikz_rule_tl
6089 \use:x { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6090 ( \l_tmpa_dim , \l_tmpb_dim ) --
6091 ( \l_@@_tmpc_dim , \l_tmpb_dim ) ;
6092 \end { tikzpicture }
6093 }

```

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).

```

6094 \cs_new_protected:Npn \@@_draw_hlines:
6095 {
6096 \int_step_inline:nnn
6097 {
6098 \bool_if:nTF { ! \g_@@_delims_bool && ! \l_@@_except_borders_bool }
6099 1 2
6100 }

```

```

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

```

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

```

6112 \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.

```

6113 \cs_set:Npn \@@_Hline_i:n #1
6114 {
6115   \peek_remove_spaces:n
6116   {
6117     \peek_meaning:NTF \Hline
6118     { \@@_Hline_ii:nn { #1 + 1 } }
6119     { \@@_Hline_iii:n { #1 } }
6120   }
6121 }
6122 \cs_set:Npn \@@_Hline_ii:nn #1 #2 { \@@_Hline_i:n { #1 } }
6123 \cs_set:Npn \@@_Hline_iii:n #1
6124 { \@@_collect_options:n { \@@_Hline_iv:nn { #1 } } }
6125 \cs_set:Npn \@@_Hline_iv:nn #1 #2
6126 {
6127   \@@_compute_rule_width:n { multiplicity = #1 , #2 }
6128   \skip_vertical:n { \l_@@_rule_width_dim }
6129   \tl_gput_right:Nx \g_@@_pre_code_after_tl
6130   {
6131     \@@_hline:n
6132     {
6133       multiplicity = #1 ,
6134       position = \int_eval:n { \c@iRow + 1 } ,
6135       total-width = \dim_use:N \l_@@_rule_width_dim ,
6136       #2
6137     }
6138   }
6139   \egroup
6140 }

```

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).

```

6141 \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`).

```

6142 \cs_new_protected:Npn \@@_custom_line:n #1
6143 {

```



```

6144 \str_clear_new:N \l_@@_command_str
6145 \str_clear_new:N \l_@@_ccommand_str
6146 \str_clear_new:N \l_@@_letter_str
6147 \tl_clear_new:N \l_@@_other_keys_tl
6148 \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.

```

6149 \bool_lazy_all:nTF
6150 {
6151   { \str_if_empty_p:N \l_@@_letter_str }
6152   { \str_if_empty_p:N \l_@@_command_str }
6153   { \str_if_empty_p:N \l_@@_ccommand_str }
6154 }
6155 { \@@_error:n { No~letter~and~no~command } }
6156 { \exp_args:NV \@@_custom_line_i:n \l_@@_other_keys_tl }
6157 }
6158 \keys_define:nn { NiceMatrix / custom-line }
6159 {
6160   letter .str_set:N = \l_@@_letter_str ,
6161   letter .value_required:n = true ,
6162   command .str_set:N = \l_@@_command_str ,
6163   command .value_required:n = true ,
6164   ccommand .str_set:N = \l_@@_ccommand_str ,
6165   ccommand .value_required:n = true ,
6166 }

```

```

6167 \cs_new_protected:Npn \@@_custom_line_i:n #1
6168 {

```

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

```

6169 \bool_set_false:N \l_@@_tikz_rule_bool
6170 \bool_set_false:N \l_@@_dotted_rule_bool
6171 \bool_set_false:N \l_@@_color_bool
6172 \keys_set:nn { NiceMatrix / custom-line-bis } { #1 }
6173 \bool_if:NT \l_@@_tikz_rule_bool
6174 {
6175   \IfPackageLoadedTF { tikz }
6176   { }
6177   { \@@_error:n { tikz~in~custom~line~without~tikz } }
6178   \bool_if:NT \l_@@_color_bool
6179   { \@@_error:n { color~in~custom~line~with~tikz } }
6180 }
6181 \bool_if:nT
6182 {
6183   \int_compare_p:nNn \l_@@_multiplicity_int > 1
6184   && \l_@@_dotted_rule_bool
6185 }
6186 { \@@_error:n { key~multiplicity~with~dotted } }
6187 \str_if_empty:NF \l_@@_letter_str
6188 {
6189   \int_compare:nTF { \str_count:N \l_@@_letter_str != 1 }
6190   { \@@_error:n { Several~letters } }
6191   {
6192     \exp_args:NnV \tl_if_in:NnTF
6193     \c_@@_forbidden_letters_str \l_@@_letter_str
6194     { \@@_error:n { Forbidden~letter } }
6195     {

```

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.

```

6196         \keys_define:nx { NiceMatrix / ColumnTypes }
6197         {
6198             \l_@@_letter_str .code:n =
6199             { \@@_v_custom_line:n { \exp_not:n { #1 } } }
6200         }
6201     }
6202 }
6203 }
6204 \str_if_empty:NF \l_@@_command_str { \@@_h_custom_line:n { #1 } }
6205 \str_if_empty:NF \l_@@_ccommand_str { \@@_c_custom_line:n { #1 } }
6206 }
6207 \str_const:Nn \c_@@_forbidden_letters_str { lcrpmbVX|()[]!@<> }

```

The previous command `\@@_custom_line_i:n` uses the following set of keys. However, the whole definition of the customized lines (as provided by the final user as argument of `custom-line`) will also be used further with other sets of keys (for instance `{NiceMatrix/Rules}`). That's why the following set of keys has some keys which are no-op.

```

6208 \keys_define:nn { NiceMatrix / custom-line-bis }
6209 {
6210     multiplicity .int_set:N = \l_@@_multiplicity_int ,
6211     multiplicity .initial:n = 1 ,
6212     multiplicity .value_required:n = true ,
6213     color .code:n = \bool_set_true:N \l_@@_color_bool ,
6214     color .value_required:n = true ,
6215     tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6216     tikz .value_required:n = true ,
6217     dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6218     dotted .value_forbidden:n = true ,
6219     total-width .code:n = { } ,
6220     total-width .value_required:n = true ,
6221     width .code:n = { } ,
6222     width .value_required:n = true ,
6223     sep-color .code:n = { } ,
6224     sep-color .value_required:n = true ,
6225     unknown .code:n = \@@_error:n { Unknown~key~for~custom~line }
6226 }

```

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

```

6227 \bool_new:N \l_@@_dotted_rule_bool
6228 \bool_new:N \l_@@_tikz_rule_bool
6229 \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`.

```

6230 \keys_define:nn { NiceMatrix / custom-line-width }
6231 {
6232     multiplicity .int_set:N = \l_@@_multiplicity_int ,
6233     multiplicity .initial:n = 1 ,
6234     multiplicity .value_required:n = true ,
6235     tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6236     total-width .code:n = \dim_set:Nn \l_@@_rule_width_dim { #1 }
6237         \bool_set_true:N \l_@@_total_width_bool ,
6238     total-width .value_required:n = true ,
6239     width .meta:n = { total-width = #1 } ,
6240     dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6241 }

```

The following command will create the command that the final user will use in its array to draw an horizontal rule (hence the ‘h’ in the name) with the full width of the array. #1 is the whole set of keys to pass to the command \@@_hline:n (which is in the internal \CodeAfter).

```
6242 \cs_new_protected:Npn \@@_h_custom_line:n #1
6243 {
```

We use \cs_set:cpn and not \cs_new:cpn because we want a local definition. Moreover, the command must *not* be protected since it begins with \noalign (which is in \Hline).

```
6244 \cs_set:cpn { nicematrix - \l_@@_command_str } { \Hline [ #1 ] }
6245 \seq_put_left:NV \l_@@_custom_line_commands_seq \l_@@_command_str
6246 }
```

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).

```
6247 \cs_new_protected:Npn \@@_c_custom_line:n #1
6248 {
```

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

```
6249 \exp_args:Nc \NewExpandableDocumentCommand
6250 { nicematrix - \l_@@_ccommand_str }
6251 { 0 { } m }
6252 {
6253 \noalign
6254 {
6255 \@@_compute_rule_width:n { #1 , ##1 }
6256 \skip_vertical:n { \l_@@_rule_width_dim }
6257 \clist_map_inline:nn
6258 { ##2 }
6259 { \@@_c_custom_line_i:nn { #1 , ##1 } { ####1 } }
6260 }
6261 }
6262 \seq_put_left:NV \l_@@_custom_line_commands_seq \l_@@_ccommand_str
6263 }
```

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*.

```
6264 \cs_new_protected:Npn \@@_c_custom_line_i:nn #1 #2
6265 {
6266 \str_if_in:nnTF { #2 } { - }
6267 { \@@_cut_on_hyphen:w #2 \q_stop }
6268 { \@@_cut_on_hyphen:w #2 - #2 \q_stop }
6269 \tl_gput_right:Nx \g_@@_pre_code_after_tl
6270 {
6271 \@@_hline:n
6272 {
6273 #1 ,
6274 start = \l_tmpa_tl ,
6275 end = \l_tmpb_tl ,
6276 position = \int_eval:n { \c@iRow + 1 } ,
6277 total-width = \dim_use:N \l_@@_rule_width_dim
6278 }
6279 }
6280 }

6281 \cs_new_protected:Npn \@@_compute_rule_width:n #1
6282 {
6283 \bool_set_false:N \l_@@_tikz_rule_bool
6284 \bool_set_false:N \l_@@_total_width_bool
6285 \bool_set_false:N \l_@@_dotted_rule_bool
6286 \keys_set_known:nn { NiceMatrix / custom-line-width } { #1 }
6287 \bool_if:NF \l_@@_total_width_bool
```

```

6288 {
6289   \bool_if:NTF \l_@@_dotted_rule_bool
6290   { \dim_set:Nn \l_@@_rule_width_dim { 2 \l_@@_xdots_radius_dim } }
6291   {
6292     \bool_if:NF \l_@@_tikz_rule_bool
6293     {
6294       \dim_set:Nn \l_@@_rule_width_dim
6295       {
6296         \arrayrulewidth * \l_@@_multiplicity_int
6297         + \doublerulesep * ( \l_@@_multiplicity_int - 1 )
6298       }
6299     }
6300   }
6301 }
6302 }
6303 \cs_new_protected:Npn \@@_v_custom_line:n #1
6304 {
6305   \@@_compute_rule_width:n { #1 }

```

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

```

6306   \tl_gput_right:Nx \g_@@_array_preamble_tl
6307   { \exp_not:N ! { \skip_horizontal:n { \dim_use:N \l_@@_rule_width_dim } } }
6308   \tl_gput_right:Nx \g_@@_pre_code_after_tl
6309   {
6310     \@@_vline:n
6311     {
6312       #1 ,
6313       position = \int_eval:n { \c@jCol + 1 } ,
6314       total-width = \dim_use:N \l_@@_rule_width_dim
6315     }
6316   }
6317 }
6318 \@@_custom_line:n
6319 { 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.

```

6320 \cs_new_protected:Npn \@@_test_hline_in_block:nnnnn #1 #2 #3 #4 #5
6321 {
6322   \bool_lazy_all:nT
6323   {
6324     { \int_compare_p:nNn \l_tmpa_tl > { #1 } }
6325     { \int_compare_p:nNn \l_tmpa_tl < { #3 + 1 } }
6326     { \int_compare_p:nNn \l_tmpb_tl > { #2 - 1 } }
6327     { \int_compare_p:nNn \l_tmpb_tl < { #4 + 1 } }
6328   }
6329   { \bool_gset_false:N \g_tmpa_bool }
6330 }

```

The same for vertical rules.

```

6331 \cs_new_protected:Npn \@@_test_vline_in_block:nnnnn #1 #2 #3 #4 #5
6332 {
6333   \bool_lazy_all:nT
6334   {
6335     { \int_compare_p:nNn \l_tmpa_tl > { #1 - 1 } }
6336     { \int_compare_p:nNn \l_tmpa_tl < { #3 + 1 } }
6337     { \int_compare_p:nNn \l_tmpb_tl > { #2 } }
6338     { \int_compare_p:nNn \l_tmpb_tl < { #4 + 1 } }
6339   }

```

```

6340     { \bool_gset_false:N \g_tmpa_bool }
6341   }
6342 \cs_new_protected:Npn \@@_test_hline_in_stroken_block:nnnn #1 #2 #3 #4
6343 {
6344   \bool_lazy_all:nT
6345   {
6346     {
6347       ( \int_compare_p:nNn \l_tmpa_tl = { #1 } )
6348       || ( \int_compare_p:nNn \l_tmpa_tl = { #3 + 1 } )
6349     }
6350     { \int_compare_p:nNn \l_tmpb_tl > { #2 - 1 } }
6351     { \int_compare_p:nNn \l_tmpb_tl < { #4 + 1 } }
6352   }
6353   { \bool_gset_false:N \g_tmpa_bool }
6354 }
6355 \cs_new_protected:Npn \@@_test_vline_in_stroken_block:nnnn #1 #2 #3 #4
6356 {
6357   \bool_lazy_all:nT
6358   {
6359     { \int_compare_p:nNn \l_tmpa_tl > { #1 - 1 } }
6360     { \int_compare_p:nNn \l_tmpa_tl < { #3 + 1 } }
6361     {
6362       ( \int_compare_p:nNn \l_tmpb_tl = { #2 } )
6363       || ( \int_compare_p:nNn \l_tmpb_tl = { #4 + 1 } )
6364     }
6365   }
6366   { \bool_gset_false:N \g_tmpa_bool }
6367 }

```

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.

```

6368 \cs_new_protected:Npn \@@_compute_corners:
6369 {

```

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.

```

6370   \seq_clear_new:N \l_@@_corners_cells_seq
6371   \clist_map_inline:Nn \l_@@_corners_clist
6372   {
6373     \str_case:nnF { ##1 }
6374     {
6375       { NW }
6376       { \@@_compute_a_corner:nnnnnn 1 1 1 1 \c@iRow \c@jCol }
6377       { NE }
6378       { \@@_compute_a_corner:nnnnnn 1 \c@jCol 1 { -1 } \c@iRow 1 }
6379       { SW }
6380       { \@@_compute_a_corner:nnnnnn \c@iRow 1 { -1 } 1 1 \c@jCol }
6381       { SE }
6382       { \@@_compute_a_corner:nnnnnn \c@iRow \c@jCol { -1 } { -1 } 1 1 }
6383     }
6384     { \@@_error:nn { bad~corner } { ##1 } }
6385   }

```

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

```

6386   \seq_if_empty:NF \l_@@_corners_cells_seq
6387   {

```

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.

```

6388     \tl_gput_right:Nx \g_@@_aux_tl
6389     {
6390         \seq_set_from_clist:Nn \exp_not:N \l_@@_corners_cells_seq
6391         { \seq_use:Nnnn \l_@@_corners_cells_seq , , , }
6392     }
6393 }
6394 }

```

“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.

```

6395 \cs_new_protected:Npn \@@_compute_a_corner:nnnnnn #1 #2 #3 #4 #5 #6
6396 {

```

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.

```

6397     \bool_set_false:N \l_tmpa_bool
6398     \int_zero_new:N \l_@@_last_empty_row_int
6399     \int_set:Nn \l_@@_last_empty_row_int { #1 }
6400     \int_step_inline:nnnn { #1 } { #3 } { #5 }
6401     {
6402         \@@_test_if_cell_in_a_block:nn { ##1 } { \int_eval:n { #2 } }
6403         \bool_lazy_or:nnTF
6404         {
6405             \cs_if_exist_p:c
6406             { pgf @ sh @ ns @ \@@_env: - ##1 - \int_eval:n { #2 } }
6407         }
6408         \l_tmpb_bool
6409         { \bool_set_true:N \l_tmpa_bool }
6410         {
6411             \bool_if:NF \l_tmpa_bool
6412             { \int_set:Nn \l_@@_last_empty_row_int { ##1 } }
6413         }
6414     }

```

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

```

6415     \bool_set_false:N \l_tmpa_bool
6416     \int_zero_new:N \l_@@_last_empty_column_int
6417     \int_set:Nn \l_@@_last_empty_column_int { #2 }
6418     \int_step_inline:nnnn { #2 } { #4 } { #6 }
6419     {
6420         \@@_test_if_cell_in_a_block:nn { \int_eval:n { #1 } } { ##1 }
6421         \bool_lazy_or:nnTF
6422         \l_tmpb_bool
6423         {
6424             \cs_if_exist_p:c
6425             { pgf @ sh @ ns @ \@@_env: - \int_eval:n { #1 } - ##1 }
6426         }

```

```

6427     { \bool_set_true:N \l_tmpa_bool }
6428     {
6429       \bool_if:NF \l_tmpa_bool
6430       { \int_set:Nn \l_@@_last_empty_column_int { ##1 } }
6431     }
6432   }

```

Now, we loop over the rows.

```

6433   \int_step_inline:nnnn { #1 } { #3 } \l_@@_last_empty_row_int
6434   {

```

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

```

6435     \bool_set_false:N \l_tmpa_bool
6436     \int_step_inline:nnnn { #2 } { #4 } \l_@@_last_empty_column_int
6437     {
6438       \@@_test_if_cell_in_a_block:nn { ##1 } { #####1 }
6439       \bool_lazy_or:nnTF
6440         \l_tmpb_bool
6441         {
6442           \cs_if_exist_p:c
6443             { pgf @ sh @ ns @ \@@_env: - ##1 - #####1 }
6444         }
6445       { \bool_set_true:N \l_tmpa_bool }
6446       {
6447         \bool_if:NF \l_tmpa_bool
6448         {
6449           \int_set:Nn \l_@@_last_empty_column_int { #####1 }
6450           \seq_put_right:Nn
6451             \l_@@_corners_cells_seq
6452             { ##1 - #####1 }
6453         }
6454       }
6455     }
6456   }
6457 }

```

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`).

```

6458 \cs_new_protected:Npn \@@_test_if_cell_in_a_block:nn #1 #2
6459 {
6460   \int_set:Nn \l_tmpa_int { #1 }
6461   \int_set:Nn \l_tmpb_int { #2 }
6462   \bool_set_false:N \l_tmpb_bool
6463   \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6464     { \@@_test_if_cell_in_block:nnnnnnn \l_tmpa_int \l_tmpb_int ##1 }
6465 }
6466 \cs_new_protected:Npn \@@_test_if_cell_in_block:nnnnnnn #1 #2 #3 #4 #5 #6 #7
6467 {
6468   \int_compare:nNnT { #3 } < { \int_eval:n { #1 + 1 } }
6469   {
6470     \int_compare:nNnT { #1 } < { \int_eval:n { #5 + 1 } }
6471     {
6472       \int_compare:nNnT { #4 } < { \int_eval:n { #2 + 1 } }
6473       {
6474         \int_compare:nNnT { #2 } < { \int_eval:n { #6 + 1 } }
6475         { \bool_set_true:N \l_tmpb_bool }
6476       }
6477     }
6478   }
6479 }

```

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.

```
6480 \bool_new:N \l_@@_block_auto_columns_width_bool
```

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

```
6481 \keys_define:nn { NiceMatrix / NiceMatrixBlock }
6482 {
6483   auto-columns-width .code:n =
6484   {
6485     \bool_set_true:N \l_@@_block_auto_columns_width_bool
6486     \dim_gzero_new:N \g_@@_max_cell_width_dim
6487     \bool_set_true:N \l_@@_auto_columns_width_bool
6488   }
6489 }

6490 \NewDocumentEnvironment { NiceMatrixBlock } { ! 0 { } }
6491 {
6492   \int_gincr:N \g_@@_NiceMatrixBlock_int
6493   \dim_zero:N \l_@@_columns_width_dim
6494   \keys_set:nn { NiceMatrix / NiceMatrixBlock } { #1 }
6495   \bool_if:NT \l_@@_block_auto_columns_width_bool
6496   {
6497     \cs_if_exist:cT
6498     { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
6499     {
6500       \exp_args:NNx \dim_set:Nn \l_@@_columns_width_dim
6501       {
6502         \use:c
6503         { @@_max_cell_width _ \int_use:N \g_@@_NiceMatrixBlock_int }
6504       }
6505     }
6506   }
6507 }
```

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).

```
6508 {
6509   \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.

```
6510   { \int_gdecr:N \g_@@_NiceMatrixBlock_int }
6511   {
6512     \bool_if:NT \l_@@_block_auto_columns_width_bool
6513     {
6514       \iow_shipout:Nn \@mainaux \ExplSyntaxOn
6515       \iow_shipout:Nx \@mainaux
6516       {
6517         \cs_gset:cpn
6518         { @@ _ max _ cell _ width _ \int_use:N \g_@@_NiceMatrixBlock_int }
6519         { \dim_eval:n { \g_@@_max_cell_width_dim + \arrayrulewidth } }
6520       }
6521       \iow_shipout:Nn \@mainaux \ExplSyntaxOff
6522     }
6523   }
6524   \ignorespacesafterend
6525 }
```


26 The extra nodes

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

```
6526 \cs_generate_variant:Nn \dim_min:nn { v n }
6527 \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).

```
6528 \cs_new_protected:Npn \@@_create_extra_nodes:
6529 {
6530   \bool_if:NTF \l_@@_medium_nodes_bool
6531   {
6532     \bool_if:NTF \l_@@_large_nodes_bool
6533     \@@_create_medium_and_large_nodes:
6534     \@@_create_medium_nodes:
6535   }
6536   { \bool_if:NT \l_@@_large_nodes_bool \@@_create_large_nodes: }
6537 }
```

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`.

```
6538 \cs_new_protected:Npn \@@_computations_for_medium_nodes:
6539 {
6540   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6541   {
6542     \dim_zero_new:c { l_@@_row_\@@_i: _min_dim }
6543     \dim_set_eq:cN { l_@@_row_\@@_i: _min_dim } \c_max_dim
6544     \dim_zero_new:c { l_@@_row_\@@_i: _max_dim }
6545     \dim_set:cn { l_@@_row_\@@_i: _max_dim } { - \c_max_dim }
6546   }
6547   \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
6548   {
6549     \dim_zero_new:c { l_@@_column_\@@_j: _min_dim }
6550     \dim_set_eq:cN { l_@@_column_\@@_j: _min_dim } \c_max_dim
6551     \dim_zero_new:c { l_@@_column_\@@_j: _max_dim }
6552     \dim_set:cn { l_@@_column_\@@_j: _max_dim } { - \c_max_dim }
6553   }
}
```

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

```
6554 \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6555 {
6556   \int_step_variable:nnNn
6557   \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.

```

6558     {
6559         \cs_if_exist:cT
6560         { 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.

```

6561     {
6562         \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { south-west }
6563         \dim_set:cn { l_@@_row_ \@@_i: _min_dim }
6564         { \dim_min:vn { l_@@_row _ \@@_i: _min_dim } \pgf@y }
6565         \seq_if_in:NxF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
6566         {
6567             \dim_set:cn { l_@@_column _ \@@_j: _min_dim }
6568             { \dim_min:vn { l_@@_column _ \@@_j: _min_dim } \pgf@x }
6569         }

```

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.

```

6570         \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { north-east }
6571         \dim_set:cn { l_@@_row _ \@@_i: _max_dim }
6572         { \dim_max:vn { l_@@_row _ \@@_i: _max_dim } \pgf@y }
6573         \seq_if_in:NxF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
6574         {
6575             \dim_set:cn { l_@@_column _ \@@_j: _max_dim }
6576             { \dim_max:vn { l_@@_column _ \@@_j: _max_dim } \pgf@x }
6577         }
6578     }
6579 }
6580 }

```

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

```

6581 \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6582 {
6583     \dim_compare:nNnT
6584     { \dim_use:c { l_@@_row _ \@@_i: _min _ dim } } = \c_max_dim
6585     {
6586         \@@_qpoint:n { row - \@@_i: - base }
6587         \dim_set:cn { l_@@_row _ \@@_i: _max _ dim } \pgf@y
6588         \dim_set:cn { l_@@_row _ \@@_i: _min _ dim } \pgf@y
6589     }
6590 }
6591 \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
6592 {
6593     \dim_compare:nNnT
6594     { \dim_use:c { l_@@_column _ \@@_j: _min _ dim } } = \c_max_dim
6595     {
6596         \@@_qpoint:n { col - \@@_j: }
6597         \dim_set:cn { l_@@_column _ \@@_j: _max _ dim } \pgf@y
6598         \dim_set:cn { l_@@_column _ \@@_j: _min _ dim } \pgf@y
6599     }
6600 }
6601 }

```

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

```

6602 \cs_new_protected:Npn \@@_create_medium_nodes:
6603 {
6604     \pgfpicture
6605     \pgfrememberpicturepositiononpagetrue
6606     \pgf@relevantforpicturesizefalse
6607     \@@_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”.

```
6608 \tl_set:Nn \l_@@_suffix_tl { -medium }
6609 \@@_create_nodes:
6610 \endpgfpicture
6611 }
```

The command `\@@_create_large_nodes:` must be used when we want to create only the “large nodes” and not the medium ones¹³. 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:`.

```
6612 \cs_new_protected:Npn \@@_create_large_nodes:
6613 {
6614 \pgfpicture
6615 \pgfrememberpicturepositiononpagetrue
6616 \pgf@relevantforpicturesizefalse
6617 \@@_computations_for_medium_nodes:
6618 \@@_computations_for_large_nodes:
6619 \tl_set:Nn \l_@@_suffix_tl { - large }
6620 \@@_create_nodes:
6621 \endpgfpicture
6622 }

6623 \cs_new_protected:Npn \@@_create_medium_and_large_nodes:
6624 {
6625 \pgfpicture
6626 \pgfrememberpicturepositiononpagetrue
6627 \pgf@relevantforpicturesizefalse
6628 \@@_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”.

```
6629 \tl_set:Nn \l_@@_suffix_tl { - medium }
6630 \@@_create_nodes:
6631 \@@_computations_for_large_nodes:
6632 \tl_set:Nn \l_@@_suffix_tl { - large }
6633 \@@_create_nodes:
6634 \endpgfpicture
6635 }
```

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.

```
6636 \cs_new_protected:Npn \@@_computations_for_large_nodes:
6637 {
6638 \int_set:Nn \l_@@_first_row_int 1
6639 \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`.

```
6640 \int_step_variable:nNn { \c@iRow - 1 } \@@_i:
6641 {
6642 \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim }
6643 {
6644 (
6645 \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } +
6646 \dim_use:c { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
6647 )
6648 / 2
6649 }
```

¹³If we want to create both, we have to use `\@@_create_medium_and_large_nodes:`

```

6650     \dim_set_eq:cc { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
6651     { l_@@_row_@@_i: _min_dim }
6652   }
6653   \int_step_variable:nNn { \c@jCol - 1 } \@@_j:
6654   {
6655     \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim }
6656     {
6657       (
6658         \dim_use:c { l_@@_column _ \@@_j: _ max _ dim } +
6659         \dim_use:c
6660         { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
6661       )
6662       / 2
6663     }
6664     \dim_set_eq:cc { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
6665     { l_@@_column _ \@@_j: _ max _ dim }
6666   }

```

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

```

6667   \dim_sub:cn
6668   { l_@@_column _ 1 _ min _ dim }
6669   \l_@@_left_margin_dim
6670   \dim_add:cn
6671   { l_@@_column _ \int_use:N \c@jCol _ max _ dim }
6672   \l_@@_right_margin_dim
6673 }

```

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).

```

6674 \cs_new_protected:Npn \@@_create_nodes:
6675 {
6676   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6677   {
6678     \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
6679     {

```

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

```

6680     \@@_pgf_rect_node:nnnnn
6681     { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
6682     { \dim_use:c { l_@@_column _ \@@_j: _min_dim } }
6683     { \dim_use:c { l_@@_row _ \@@_i: _min_dim } }
6684     { \dim_use:c { l_@@_column _ \@@_j: _max_dim } }
6685     { \dim_use:c { l_@@_row _ \@@_i: _max_dim } }
6686     \str_if_empty:NF \l_@@_name_str
6687     {
6688       \pgfnodealias
6689       { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
6690       { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
6691     }
6692   }
6693 }

```

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 .

```

6694   \seq_map_pairwise_function:NNN
6695   \g_@@_multicolumn_cells_seq
6696   \g_@@_multicolumn_sizes_seq
6697   \@@_node_for_multicolumn:nn
6698 }

```

```

6699 \cs_new_protected:Npn \@@_extract_coords_values: #1 - #2 \q_stop
6700 {
6701   \cs_set_nopar:Npn \@@_i: { #1 }
6702   \cs_set_nopar:Npn \@@_j: { #2 }
6703 }

```

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”).

```

6704 \cs_new_protected:Npn \@@_node_for_multicolumn:nn #1 #2
6705 {
6706   \@@_extract_coords_values: #1 \q_stop
6707   \@@_pgf_rect_node:nnnnn
6708   { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
6709   { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } }
6710   { \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } }
6711   { \dim_use:c { l_@@_column _ \int_eval:n { \@@_j: +#2-1 } _ max _ dim } }
6712   { \dim_use:c { l_@@_row _ \@@_i: _ max _ dim } }
6713   \str_if_empty:NF \l_@@_name_str
6714   {
6715     \pgfnodealias
6716     { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
6717     { \int_use:N \g_@@_env_int - \@@_i: - \@@_j: \l_@@_suffix_tl}
6718   }
6719 }

```

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.

```

6720 \keys_define:nn { NiceMatrix / Block / FirstPass }
6721 {
6722   l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
6723   l .value_forbidden:n = true ,
6724   r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
6725   r .value_forbidden:n = true ,
6726   c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
6727   c .value_forbidden:n = true ,
6728   L .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
6729   L .value_forbidden:n = true ,
6730   R .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
6731   R .value_forbidden:n = true ,
6732   C .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
6733   C .value_forbidden:n = true ,
6734   t .code:n = \str_set:Nn \l_@@_vpos_of_block_str t ,
6735   t .value_forbidden:n = true ,
6736   T .code:n = \str_set:Nn \l_@@_vpos_of_block_str T ,
6737   T .value_forbidden:n = true ,
6738   b .code:n = \str_set:Nn \l_@@_vpos_of_block_str b ,
6739   b .value_forbidden:n = true ,
6740   B .code:n = \str_set:Nn \l_@@_vpos_of_block_str B ,
6741   B .value_forbidden:n = true ,
6742   color .code:n =
6743     \@@_color:n { #1 }
6744     \tl_set_rescan:Nnn
6745     \l_@@_draw_tl

```

```

6746     { \char_set_catcode_other:N ! }
6747     { #1 } ,
6748     color .value_required:n = true ,
6749     respect-arraystretch .bool_set:N = \l_@@_respect_arraystretch_bool ,
6750     respect-arraystretch .default:n = true
6751 }

```

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.

```

6752 \cs_new_protected:Npn \@@_Block: { \@@_collect_options:n { \@@_Block_i: } }

```

```

6753 \NewExpandableDocumentCommand \@@_Block_i: { m m D < > { } +m }
6754 {

```

If the first mandatory argument of the command (which is the size of the block with the syntax $i-j$) has not been provided by the user, you use 1-1 (that is to say a block of only one cell).

```

6755     \peek_remove_spaces:n
6756     {
6757         \tl_if_blank:nTF { #2 }
6758         { \@@_Block_i 1-1 \q_stop }
6759         {
6760             \int_compare:nNnTF { \char_value_catcode:n { 45 } } = { 13 }
6761             \@@_Block_i_czech \@@_Block_i
6762             #2 \q_stop
6763         }
6764         { #1 } { #3 } { #4 }
6765     }
6766 }

```

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

```

6767 \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`.

```

6768 {
6769     \char_set_catcode_active:N -
6770     \cs_new:Npn \@@_Block_i_czech #1-#2 \q_stop { \@@_Block_ii:nnnnn { #1 } { #2 } }
6771 }

```

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.

```

6772 \cs_new_protected:Npn \@@_Block_ii:nnnnn #1 #2 #3 #4 #5
6773 {

```

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).

```

6774     \bool_lazy_or:nnTF
6775     { \tl_if_blank_p:n { #1 } }
6776     { \str_if_eq_p:nn { #1 } { * } }
6777     { \int_set:Nn \l_tmpa_int { 100 } }
6778     { \int_set:Nn \l_tmpa_int { #1 } }
6779     \bool_lazy_or:nnTF
6780     { \tl_if_blank_p:n { #2 } }
6781     { \str_if_eq_p:nn { #2 } { * } }
6782     { \int_set:Nn \l_tmpb_int { 100 } }
6783     { \int_set:Nn \l_tmpb_int { #2 } }

```

If the block is mono-column.

```

6784 \int_compare:nNnTF \l_tmpb_int = 1
6785 {
6786   \str_if_empty:NTF \l_@@_hpos_cell_str
6787     { \str_set:Nn \l_@@_hpos_block_str c }
6788     { \str_set_eq:NN \l_@@_hpos_block_str \l_@@_hpos_cell_str }
6789 }
6790 { \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.

```

6791 \keys_set_known:nn { NiceMatrix / Block / FirstPass } { #3 }
6792 \tl_set:Nx \l_tmpa_tl
6793 {
6794   { \int_use:N \c@iRow }
6795   { \int_use:N \c@jCol }
6796   { \int_eval:n { \c@iRow + \l_tmpa_int - 1 } }
6797   { \int_eval:n { \c@jCol + \l_tmpb_int - 1 } }
6798 }

```

Now, `\l_tmpa_tl` contains an “object” corresponding to the position of the block with four components, each of them surrounded by curly brackets:

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

If the block is mono-column or mono-row, we have a special treatment. That’s why we have two macros: `\@@_Block_iv:nnnnn` and `\@@_Block_v:nnnnn` (the five arguments of those macros are provided by curryfication).

```

6799 \bool_if:nTF
6800 {
6801   (
6802     \int_compare_p:nNn { \l_tmpa_int } = 1
6803     ||
6804     \int_compare_p:nNn { \l_tmpb_int } = 1
6805   )
6806   && ! \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.

```

6807   && ! \l_@@_X_column_bool
6808 }
6809 { \exp_args:Nxx \@@_Block_iv:nnnnn }
6810 { \exp_args:Nxx \@@_Block_v:nnnnn }
6811 { \l_tmpa_int } { \l_tmpb_int } { #3 } { #4 } { #5 }
6812 }

```

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.

```

6813 \cs_new_protected:Npn \@@_Block_iv:nnnnn #1 #2 #3 #4 #5
6814 {
6815   \int_gincr:N \g_@@_block_box_int
6816   \cs_set_protected_nopar:Npn \diagbox ##1 ##2

```

```

6817 {
6818   \tl_gput_right:Nx \g_@@_pre_code_after_tl
6819   {
6820     \@@_actually_diagbox:nnnnnn
6821     { \int_use:N \c@iRow }
6822     { \int_use:N \c@jCol }
6823     { \int_eval:n { \c@iRow + #1 - 1 } }
6824     { \int_eval:n { \c@jCol + #2 - 1 } }
6825     { \exp_not:n { ##1 } } { \exp_not:n { ##2 } }
6826   }
6827 }
6828 \box_gclear_new:c
6829 { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }

```

Now, we will actually compose the content of the `\Block` in a TeX box. *Be careful:* if after, the construction of the box, the boolean `\g_@@_rotate_bool` is raised (which means that the command `\rotate` was present in the content of the `\Block`) we will rotate the box but also, maybe, change the position of the baseline!

```

6830 \hbox_gset:cn
6831 { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
6832 {

```

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}`).

```

6833   \tl_if_empty:NTF \l_@@_color_tl
6834   { \int_compare:nNnT { #2 } = 1 \set@color }
6835   { \@@_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`.

```

6836   \int_compare:nNnT { #1 } = 1
6837   {
6838     \int_compare:nNnTF \c@iRow = 0
6839     \l_@@_code_for_first_row_tl
6840     {
6841       \int_compare:nNnT \c@iRow = \l_@@_last_row_int
6842       \l_@@_code_for_last_row_tl
6843     }
6844     \g_@@_row_style_tl
6845   }
6846   \bool_if:NF \l_@@_respect_arraystretch_bool
6847   { \cs_set:Npn \arraystretch { 1 } }
6848   \dim_zero:N \extrarowheight

```

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

```

6849   #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.).

```

6850   \@@_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}`.

```

6851   \bool_if:NTF \l_@@_tabular_bool
6852   {
6853     \bool_lazy_all:nTF
6854     {
6855       { \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.

```
6856         { \dim_compare:n { \l_@@_col_width_dim >= \c_zero_dim } }
6857         { ! \g_@@_rotate_bool }
6858     }
```

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

```
6859     {
6860         \use:x
6861         {
6862             \exp_not:N \begin { minipage }%
6863             [ \str_lowercase:V { \l_@@_vpos_of_block_str } ]
6864             { \l_@@_col_width_dim }
6865             \str_case:Vn \l_@@_hpos_block_str
6866             { c \centering r \raggedleft l \raggedright }
6867         }
6868         #5
6869         \end { minipage }
6870     }
```

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

```
6871     {
6872         \use:x
6873         {
6874             \exp_not:N \begin { tabular }%
6875             [ \str_lowercase:V { \l_@@_vpos_of_block_str } ]
6876             { @ { } \l_@@_hpos_block_str @ { } }
6877         }
6878         #5
6879         \end { tabular }
6880     }
6881 }
```

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

```
6882     {
6883         \c_math_toggle_token
6884         \use:x
6885         {
6886             \exp_not:N \begin { array }%
6887             [ \str_lowercase:V { \l_@@_vpos_of_block_str } ]
6888             { @ { } \l_@@_hpos_block_str @ { } }
6889         }
6890         #5
6891         \end { array }
6892         \c_math_toggle_token
6893     }
6894 }
```

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).

```
6895     \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.

```
6896     \int_compare:nNnT { #2 } = 1
6897     {
6898         \dim_gset:Nn \g_@@_blocks_wd_dim
6899         {
6900             \dim_max:nn
6901             \g_@@_blocks_wd_dim
6902             {
6903                 \box_wd:c
```

```

6904         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
6905     }
6906 }
6907 }

```

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.

```

6908 \int_compare:nNnT { #1 } = 1
6909 {
6910     \dim_gset:Nn \g_@@_blocks_ht_dim
6911     {
6912         \dim_max:nn
6913         \g_@@_blocks_ht_dim
6914         {
6915             \box_ht:c
6916             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
6917         }
6918     }
6919     \dim_gset:Nn \g_@@_blocks_dp_dim
6920     {
6921         \dim_max:nn
6922         \g_@@_blocks_dp_dim
6923         {
6924             \box_dp:c
6925             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
6926         }
6927     }
6928 }
6929 \seq_gput_right:Nx \g_@@_blocks_seq
6930 {
6931     \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.

```

6932     {
6933         \exp_not:n { #3 } ,
6934         \l_@@_hpos_block_str ,

```

Now, we put a key for the vertical alignment.

```

6935     \bool_if:NT \g_@@_rotate_bool
6936     {
6937         \bool_if:NTF \g_@@_rotate_c_bool
6938         { v-center }
6939         { \int_compare:nNnT \c@iRow = \l_@@_last_row_int T }
6940     }
6941 }
6942 {
6943     \box_use_drop:c
6944     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
6945 }
6946 }
6947 }
6948 \bool_set_false:N \g_@@_rotate_c_bool
6949 }

```

```

6950 \cs_new:Npn \@@_adjust_hpos_rotate:
6951 {
6952     \bool_if:NT \g_@@_rotate_bool
6953     {
6954         \str_set:Nx \l_@@_hpos_block_str
6955         {
6956             \bool_if:NTF \g_@@_rotate_c_bool

```

```

6957         { c }
6958     {
6959         \str_case:VnF \l_@@_vpos_of_block_str
6960         { b l B l t r T r }
6961         { \int_compare:nNnTF \c@iRow = \l_@@_last_row_int r l }
6962     }
6963 }
6964 }
6965 }

```

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

```

6966 \cs_new_protected:Npn \@@_rotate_box_of_block:
6967 {
6968     \box_grotate:cn
6969     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
6970     { 90 }
6971     \int_compare:nNnT \c@iRow = \l_@@_last_row_int
6972     {
6973         \vbox_gset_top:cn
6974         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
6975         {
6976             \skip_vertical:n { 0.8 ex }
6977             \box_use:c
6978             { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
6979         }
6980     }
6981     \bool_if:NT \g_@@_rotate_c_bool
6982     {
6983         \hbox_gset:cn
6984         { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
6985         {
6986             \c_math_toggle_token
6987             \vcenter
6988             {
6989                 \box_use:c
6990                 { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
6991             }
6992             \c_math_toggle_token
6993         }
6994     }
6995 }

```

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.

```

6996 \cs_new_protected:Npn \@@_Block_v:nnnnn #1 #2 #3 #4 #5
6997 {
6998     \seq_gput_right:Nx \g_@@_blocks_seq
6999     {
7000         \l_tmpa_tl
7001         { \exp_not:n { #3 } }
7002         {
7003             \bool_if:NTF \l_@@_tabular_bool
7004             {
7005                 \group_begin:
7006                 \bool_if:NF \l_@@_respect_arraystretch_bool
7007                 { \cs_set:Npn \exp_not:N \arraystretch { 1 } }

```

```

7008         \exp_not:n
7009         {
7010             \dim_zero:N \extrarowheight
7011             #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).

```

7012         % \@@_adjust_hpos_rotate:
7013         \use:x
7014         {
7015             \exp_not:N \begin { tabular } [ \l_@@_vpos_of_block_str ]
7016             { @ { } \l_@@_hpos_block_str @ { } }
7017         }
7018         #5
7019         \end { tabular }
7020     }
7021     \group_end:
7022 }

```

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

```

7023     {
7024         \group_begin:
7025         \bool_if:NF \l_@@_respect_arraystretch_bool
7026         { \cs_set:Npn \exp_not:N \arraystretch { 1 } }
7027         \exp_not:n
7028         {
7029             \dim_zero:N \extrarowheight
7030             #4
7031             % \@@_adjust_hpos_rotate:
7032             \c_math_toggle_token      :n c
7033             \use:x
7034             {
7035                 \exp_not:N \begin { array } [ \l_@@_vpos_of_block_str ]
7036                 { @ { } \l_@@_hpos_block_str @ { } }
7037             }
7038             #5
7039             \end { array }
7040             \c_math_toggle_token
7041         }
7042         \group_end:
7043     }
7044 }
7045 }
7046 }

```

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).

```

7047 \keys_define:nn { NiceMatrix / Block / SecondPass }
7048 {
7049     tikz .code:n =
7050         \IfPackageLoadedTF { tikz }
7051         { \seq_put_right:Nn \l_@@_tikz_seq { { #1 } } }
7052         { \@@_error:n { tikz-key-without-tikz } } ,
7053     tikz .value_required:n = true ,
7054     fill .code:n =
7055         \tl_set_rescan:Nnn
7056         \l_@@_fill_tl
7057         { \char_set_catcode_other:N ! }
7058         { #1 } ,

```

```

7059 fill .value_required:n = true ,
7060 opacity .tl_set:N = \l_@@_opacity_tl ,
7061 opacity .value_required:n = true ,
7062 draw .code:n =
7063   \tl_set_rescan:Nnn
7064     \l_@@_draw_tl
7065     { \char_set_catcode_other:N ! }
7066     { #1 } ,
7067 draw .default:n = default ,
7068 rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
7069 rounded-corners .default:n = 4 pt ,
7070 color .code:n =
7071   \@@_color:n { #1 }
7072   \tl_set_rescan:Nnn
7073     \l_@@_draw_tl
7074     { \char_set_catcode_other:N ! }
7075     { #1 } ,
7076 borders .clist_set:N = \l_@@_borders_clist ,
7077 borders .value_required:n = true ,
7078 hvlines .meta:n = { vlines , hlines } ,
7079 vlines .bool_set:N = \l_@@_vlines_block_bool ,
7080 vlines .default:n = true ,
7081 hlines .bool_set:N = \l_@@_hlines_block_bool ,
7082 hlines .default:n = true ,
7083 line-width .dim_set:N = \l_@@_line_width_dim ,
7084 line-width .value_required:n = true ,

```

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

```

7085 l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7086 r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7087 c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7088 L .code:n = \str_set:Nn \l_@@_hpos_block_str l
7089   \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7090 R .code:n = \str_set:Nn \l_@@_hpos_block_str r
7091   \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7092 C .code:n = \str_set:Nn \l_@@_hpos_block_str c
7093   \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7094 t .code:n = \str_set:Nn \l_@@_vpos_of_block_str t ,
7095 T .code:n = \str_set:Nn \l_@@_vpos_of_block_str T ,
7096 b .code:n = \str_set:Nn \l_@@_vpos_of_block_str b ,
7097 B .code:n = \str_set:Nn \l_@@_vpos_of_block_str B ,
7098 v-center .code:n = \str_set:Nn \l_@@_vpos_of_block_str { c } ,
7099 v-center .value_forbidden:n = true ,
7100 name .tl_set:N = \l_@@_block_name_str ,
7101 name .value_required:n = true ,
7102 name .initial:n = ,
7103 respect-arraystretch .bool_set:N = \l_@@_respect_arraystretch_bool ,
7104 transparent .bool_set:N = \l_@@_transparent_bool ,
7105 transparent .default:n = true ,
7106 transparent .initial:n = false ,
7107 unknown .code:n = \@@_error:n { Unknown-key-for-Block }
7108 }

```

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.

```

7109 \cs_new_protected:Npn \@@_draw_blocks:
7110 {
7111   \cs_set_eq:NN \ialign \@@_old_ialign:
7112   \seq_map_inline:Nn \g_@@_blocks_seq { \@@_Block_iv:nnnnnn ##1 }
7113 }
7114 \cs_new_protected:Npn \@@_Block_iv:nnnnnn #1 #2 #3 #4 #5 #6
7115 {

```

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

```
7116 \int_zero_new:N \l_@@_last_row_int
7117 \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.

```
7118 \int_compare:nNnTF { #3 } > { 99 }
7119 { \int_set_eq:NN \l_@@_last_row_int \c@iRow }
7120 { \int_set:Nn \l_@@_last_row_int { #3 } }
7121 \int_compare:nNnTF { #4 } > { 99 }
7122 { \int_set_eq:NN \l_@@_last_col_int \c@jCol }
7123 { \int_set:Nn \l_@@_last_col_int { #4 } }
7124 \int_compare:nNnTF \l_@@_last_col_int > \g_@@_col_total_int
7125 {
7126   \int_compare:nTF
7127   { \l_@@_last_col_int <= \g_@@_static_num_of_col_int }
7128   {
7129     \msg_error:nnnn { nicematrix } { Block-too-large~2 } { #1 } { #2 }
7130     @@_msg_redirect_name:nn { Block-too-large~2 } { none }
7131     @@_msg_redirect_name:nn { columns-not-used } { none }
7132   }
7133   { \msg_error:nnnn { nicematrix } { Block-too-large~1 } { #1 } { #2 } }
7134 }
7135 {
7136   \int_compare:nNnTF \l_@@_last_row_int > \g_@@_row_total_int
7137   { \msg_error:nnnn { nicematrix } { Block-too-large~1 } { #1 } { #2 } }
7138   { @@_Block_v:nnnnnn { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } }
7139 }
7140 }
```

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

```
7141 \cs_new_protected:Npn @@_Block_v:nnnnnn #1 #2 #3 #4 #5 #6
7142 {
```

The group is for the keys.

```
7143 \group_begin:
7144 \int_compare:nNnT { #1 } = { #3 }
7145 { \str_set:Nn \l_@@_vpos_of_block_str { t } }
7146 \keys_set:nn { NiceMatrix / Block / SecondPass } { #5 }
7147 \bool_if:NT \l_@@_vlines_block_bool
7148 {
7149   \tl_gput_right:Nx \g_nicematrix_code_after_tl
7150   {
7151     @@_vlines_block:nnn
7152     { \exp_not:n { #5 } }
7153     { #1 - #2 }
7154     { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7155   }
7156 }
7157 \bool_if:NT \l_@@_hlines_block_bool
7158 {
7159   \tl_gput_right:Nx \g_nicematrix_code_after_tl
7160   {
7161     @@_hlines_block:nnn
7162     { \exp_not:n { #5 } }
7163     { #1 - #2 }
```

```

7164         { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7165     }
7166 }
7167 \bool_if:nF
7168 {
7169     \l_@@_transparent_bool
7170     || ( \l_@@_vlines_block_bool && \l_@@_hlines_block_bool )
7171 }
7172 {

```

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).

```

7173     \seq_gput_left:Nx \g_@@_pos_of_blocks_seq
7174     { { #1 } { #2 } { #3 } { #4 } { \l_@@_block_name_str } }
7175 }

7176 \bool_lazy_and:nnT
7177 { ! (\tl_if_empty_p:N \l_@@_draw_tl) }
7178 { \l_@@_hlines_block_bool || \l_@@_vlines_block_bool }
7179 { \@@_error:n { hlines-with-color } }

7180 \tl_if_empty:NF \l_@@_draw_tl
7181 {
7182     \tl_gput_right:Nx \g_nicematrix_code_after_tl
7183     {
7184         \@@_stroke_block:nnn
7185         { \exp_not:n { #5 } } % #5 are the options
7186         { #1 - #2 }
7187         { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7188     }
7189     \seq_gput_right:Nn \g_@@_pos_of_stroken_blocks_seq
7190     { { #1 } { #2 } { #3 } { #4 } }
7191 }

7192 \clist_if_empty:NF \l_@@_borders_clist
7193 {
7194     \tl_gput_right:Nx \g_nicematrix_code_after_tl
7195     {
7196         \@@_stroke_borders_block:nnn
7197         { \exp_not:n { #5 } }
7198         { #1 - #2 }
7199         { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7200     }
7201 }

7202 \tl_if_empty:NF \l_@@_fill_tl
7203 {
7204     \tl_if_empty:NF \l_@@_opacity_tl
7205     {
7206         \tl_if_head_eq_meaning:nNTF \l_@@_fill_tl [
7207             {
7208                 \tl_set:Nx \l_@@_fill_tl
7209                 {
7210                     [ opacity = \l_@@_opacity_tl ,
7211                     \tl_tail:V \l_@@_fill_tl
7212                 }
7213             }
7214             {
7215                 \tl_set:Nx \l_@@_fill_tl
7216                 { [ opacity = \l_@@_opacity_tl ] { \l_@@_fill_tl } }
7217             }
7218         }
7219         \tl_gput_right:Nx \g_@@_pre_code_before_tl

```

```

7220     {
7221         \exp_not:N \roundedrectanglecolor
7222         \exp_args:NV \tl_if_head_eq_meaning:nNTF \l_@@_fill_tl [
7223             { \l_@@_fill_tl }
7224             { { \l_@@_fill_tl } }
7225             { #1 - #2 }
7226             { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7227             { \dim_use:N \l_@@_rounded_corners_dim }
7228         }
7229     }
7230 \seq_if_empty:NF \l_@@_tikz_seq
7231 {
7232     \tl_gput_right:Nx \g_nicematrix_code_before_tl
7233     {
7234         \@@_block_tikz:nnnnn
7235         { #1 }
7236         { #2 }
7237         { \int_use:N \l_@@_last_row_int }
7238         { \int_use:N \l_@@_last_col_int }
7239         { \seq_use:Nn \l_@@_tikz_seq { , } }
7240     }
7241 }
7242 \cs_set_protected_nopar:Npn \diagbox ##1 ##2
7243 {
7244     \tl_gput_right:Nx \g_@@_pre_code_after_tl
7245     {
7246         \@@_actually_diagbox:nnnnnn
7247         { #1 }
7248         { #2 }
7249         { \int_use:N \l_@@_last_row_int }
7250         { \int_use:N \l_@@_last_col_int }
7251         { \exp_not:n { ##1 } } { \exp_not:n { ##2 } }
7252     }
7253 }
7254 \hbox_set:Nn \l_@@_cell_box { \set@color #6 }
7255 \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.

```

7256 \pgfpicture
7257 \pgfrememberpicturepositiononpagetrue

```



```

7258 \pgf@relevantforpicturesizefalse
7259 \@@_qpoint:n { row - #1 }
7260 \dim_set_eq:NN \l_tmpa_dim \pgf@y
7261 \@@_qpoint:n { col - #2 }
7262 \dim_set_eq:NN \l_tmpb_dim \pgf@x
7263 \@@_qpoint:n { row - \int_eval:n { \l_@@_last_row_int + 1 } }
7264 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7265 \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7266 \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.

```

7267 \@@_pgf_rect_node:nnnnn
7268 { \@@_env: - #1 - #2 - block }
7269 \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
7270 \str_if_empty:NF \l_@@_block_name_str
7271 {
7272 \pgfnodealias
7273 { \@@_env: - \l_@@_block_name_str }
7274 { \@@_env: - #1 - #2 - block }
7275 \str_if_empty:NF \l_@@_name_str
7276 {
7277 \pgfnodealias
7278 { \l_@@_name_str - \l_@@_block_name_str }
7279 { \@@_env: - #1 - #2 - block }
7280 }
7281 }

```

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.

```

7282 \bool_if:NF \l_@@_hpos_of_block_cap_bool
7283 {
7284 \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.

```

7285 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
7286 {

```

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.

```

7287 \cs_if_exist:cT
7288 { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
7289 {
7290 \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
7291 {
7292 \pgfpointanchor { \@@_env: - ##1 - #2 } { west }
7293 \dim_set:Nn \l_tmpb_dim { \dim_min:nn \l_tmpb_dim \pgf@x }
7294 }
7295 }
7296 }

```

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.

```

7297 \dim_compare:nNnT \l_tmpb_dim = \c_max_dim
7298 {
7299 \@@_qpoint:n { col - #2 }
7300 \dim_set_eq:NN \l_tmpb_dim \pgf@x
7301 }
7302 \dim_set:Nn \l_@@_tmpd_dim { - \c_max_dim }
7303 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int

```

```

7304     {
7305         \cs_if_exist:cT
7306         { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
7307         {
7308             \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
7309             {
7310                 \pgfpointanchor
7311                 { \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
7312                 { east }
7313                 \dim_set:Nn \l_@@_tmpd_dim { \dim_max:nn \l_@@_tmpd_dim \pgf@x }
7314             }
7315         }
7316     }
7317     \dim_compare:nNnT \l_@@_tmpd_dim = { - \c_max_dim }
7318     {
7319         \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7320         \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
7321     }
7322     \@@_pgf_rect_node:nnnnn
7323     { \@@_env: - #1 - #2 - block - short }
7324     \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
7325 }

```

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.

```

7326     \bool_if:NT \l_@@_medium_nodes_bool
7327     {
7328         \@@_pgf_rect_node:nnn
7329         { \@@_env: - #1 - #2 - block - medium }
7330         { \pgfpointanchor { \@@_env: - #1 - #2 - medium } { north-west } }
7331         {
7332             \pgfpointanchor
7333             { \@@_env:
7334               - \int_use:N \l_@@_last_row_int
7335               - \int_use:N \l_@@_last_col_int - medium
7336             }
7337             { south-east }
7338         }
7339     }

```

Now, we will put the label of the block.

```

7340     \bool_lazy_any:nTF
7341     {
7342         { \str_if_eq_p:Vn \l_@@_vpos_of_block_str { c } }
7343         { \str_if_eq_p:Vn \l_@@_vpos_of_block_str { T } }
7344         { \str_if_eq_p:Vn \l_@@_vpos_of_block_str { B } }
7345     }
7346     {

```

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

```

7347         \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`.

```

7348         \bool_if:nT \g_@@_last_col_found_bool
7349         {
7350             \int_compare:nNnT { #2 } = \g_@@_col_total_int
7351             { \str_set:Nn \l_@@_hpos_block_str l }
7352         }

```

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

```

7353         \tl_set:Nx \l_tmpa_tl
7354         {
7355             \str_case:Vn \l_@@_vpos_of_block_str

```

```

7356     {
7357         c {
7358             \str_case:Vn \l_@@_hpos_block_str
7359             {
7360                 c { center }
7361                 l { west }
7362                 r { east }
7363             }
7364         }
7365     }
7366     T {
7367         \str_case:Vn \l_@@_hpos_block_str
7368         {
7369             c { north }
7370             l { north-west }
7371             r { north-east }
7372         }
7373     }
7374 }
7375 B {
7376     \str_case:Vn \l_@@_hpos_block_str
7377     {
7378         c { south }
7379         l { south-west }
7380         r { south-east }
7381     }
7382 }
7383 }
7384 }
7385 }
7386 \pgftransformshift
7387 {
7388     \pgfpointanchor
7389     {
7390         \@@_env: - #1 - #2 - block
7391         \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
7392     }
7393     { \l_tmpa_tl }
7394 }
7395 \pgfset
7396 {
7397     inner~xsep = \c_zero_dim ,
7398     inner~ysep = \l_@@_block_ysep_dim
7399 }
7400 \pgfnode
7401 { rectangle }
7402 { \l_tmpa_tl }
7403 { \box_use_drop:N \l_@@_cell_box } { } { } { }
7404 }

```

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

```

7405 {
7406     \pgfextracty \l_tmpa_dim
7407     {
7408         \@@_qpoint:n
7409         {
7410             row - \str_if_eq:VnTF \l_@@_vpos_of_block_str { b } { #3 } { #1 }
7411             - base
7412         }
7413     }
7414     \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.

```

7415 \pgfpointanchor
7416 {
7417   \@@_env: - #1 - #2 - block
7418   \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
7419 }
7420 {
7421   \str_case:Vn \l_@@_hpos_block_str
7422   {
7423     c { center }
7424     l { west }
7425     r { east }
7426   }
7427 }

```

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

```

7428 \pgftransformshift { \pgfpoint \pgf@x \l_tmpa_dim }
7429 \pgfset { inner-sep = \c_zero_dim }
7430 \pgfnode
7431 { rectangle }
7432 {
7433   \str_case:Vn \l_@@_hpos_block_str
7434   {
7435     c { base }
7436     l { base~west }
7437     r { base~east }
7438   }
7439 }
7440 { \box_use_drop:N \l_@@_cell_box } { } { }
7441 }
7442 \endpgfpicture
7443 \group_end:
7444 }

```

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).

```

7445 \cs_new_protected:Npn \@@_stroke_block:nnn #1 #2 #3
7446 {
7447   \group_begin:
7448   \tl_clear:N \l_@@_draw_tl
7449   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
7450   \keys_set_known:nn { NiceMatrix / BlockStroke } { #1 }
7451   \pgfpicture
7452   \pgfrememberpicturepositiononpagetrue
7453   \pgf@relevantforpicturesizefalse
7454   \tl_if_empty:NF \l_@@_draw_tl
7455   {

```

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

```

7456   \str_if_eq:VnTF \l_@@_draw_tl { default }
7457   { \CT@arc@ }
7458   { \@@_color:V \l_@@_draw_tl }
7459 }
7460 \pgfsetcornersarced
7461 {
7462   \pgfpoint
7463   { \l_@@_rounded_corners_dim }
7464   { \l_@@_rounded_corners_dim }
7465 }
7466 \@@_cut_on_hyphen:w #2 \q_stop
7467 \bool_lazy_and:nnT
7468 { \int_compare_p:n { \l_tmpa_tl <= \c@iRow } }

```

```

7469 { \int_compare_p:n { \l_tmpb_tl <= \c@jCol } }
7470 {
7471   \@@_qpoint:n { row - \l_tmpa_tl }
7472   \dim_set_eq:NN \l_tmpb_dim \pgf@y
7473   \@@_qpoint:n { col - \l_tmpb_tl }
7474   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
7475   \@@_cut_on_hyphen:w #3 \q_stop
7476   \int_compare:nNnT \l_tmpa_tl > \c@iRow
7477     { \tl_set:Nx \l_tmpa_tl { \int_use:N \c@iRow } }
7478   \int_compare:nNnT \l_tmpb_tl > \c@jCol
7479     { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@jCol } }
7480   \@@_qpoint:n { row - \int_eval:n { \l_tmpa_tl + 1 } }
7481   \dim_set_eq:NN \l_tmpa_dim \pgf@y
7482   \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
7483   \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
7484   \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
7485   \pgfpathrectanglecorners
7486     { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
7487     { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
7488   \dim_compare:nNnTF \l_@@_rounded_corners_dim = \c_zero_dim
7489     { \pgfusepathqstroke }
7490     { \pgfusepath { stroke } }
7491 }
7492 \endpgfpicture
7493 \group_end:
7494 }

```

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

```

7495 \keys_define:nn { NiceMatrix / BlockStroke }
7496 {
7497   color .tl_set:N = \l_@@_draw_tl ,
7498   draw .code:n =
7499     \exp_args:Nx \tl_if_empty:nF { #1 } { \tl_set:Nn \l_@@_draw_tl { #1 } } ,
7500   draw .default:n = default ,
7501   line-width .dim_set:N = \l_@@_line_width_dim ,
7502   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
7503   rounded-corners .default:n = 4 pt
7504 }

```

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).

```

7505 \cs_new_protected:Npn \@@_vlines_block:nnn #1 #2 #3
7506 {
7507   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
7508   \keys_set_known:nn { NiceMatrix / BlockBorders } { #1 }
7509   \@@_cut_on_hyphen:w #2 \q_stop
7510   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
7511   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
7512   \@@_cut_on_hyphen:w #3 \q_stop
7513   \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
7514   \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
7515   \int_step_inline:nnn \l_@@_tmpd_tl \l_tmpb_tl
7516   {
7517     \use:x
7518     {
7519       \@@_vline:n
7520       {
7521         position = ##1 ,
7522         start = \l_@@_tmpc_tl ,
7523         end = \int_eval:n { \l_tmpa_tl - 1 } ,
7524         total-width = \dim_use:N \l_@@_line_width_dim % added 2022-08-06
7525       }

```

```

7526     }
7527   }
7528 }
7529 \cs_new_protected:Npn \@@_hlines_block:nnn #1 #2 #3
7530 {
7531   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
7532   \keys_set_known:nn { NiceMatrix / BlockBorders } { #1 }
7533   \@@_cut_on_hyphen:w #2 \q_stop
7534   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
7535   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
7536   \@@_cut_on_hyphen:w #3 \q_stop
7537   \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
7538   \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
7539   \int_step_inline:nnn \l_@@_tmpc_tl \l_tmpa_tl
7540   {
7541     \use:x
7542     {
7543       \@@_hline:n
7544       {
7545         position = ##1 ,
7546         start = \l_@@_tmpd_tl ,
7547         end = \int_eval:n { \l_tmpb_tl - 1 } ,
7548         total-width = \dim_use:N \l_@@_line_width_dim
7549       }
7550     }
7551   }
7552 }

```

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).

```

7553 \cs_new_protected:Npn \@@_stroke_borders_block:nnn #1 #2 #3
7554 {
7555   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
7556   \keys_set_known:nn { NiceMatrix / BlockBorders } { #1 }
7557   \dim_compare:nNnTF \l_@@_rounded_corners_dim > \c_zero_dim
7558   { \@@_error:n { borders~forbidden } }
7559   {
7560     \tl_clear_new:N \l_@@_borders_tikz_tl
7561     \keys_set:nV
7562     { NiceMatrix / OnlyForTikzInBorders }
7563     \l_@@_borders_clist
7564     \@@_cut_on_hyphen:w #2 \q_stop
7565     \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
7566     \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
7567     \@@_cut_on_hyphen:w #3 \q_stop
7568     \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
7569     \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
7570     \@@_stroke_borders_block_i:
7571   }
7572 }
7573 \hook_gput_code:nnn { begindocument } { . }
7574 {
7575   \cs_new_protected:Npx \@@_stroke_borders_block_i:
7576   {
7577     \c_@@_pgfortikzpicture_tl
7578     \@@_stroke_borders_block_ii:
7579     \c_@@_endpgfortikzpicture_tl
7580   }
7581 }
7582 \cs_new_protected:Npn \@@_stroke_borders_block_ii:
7583 {

```

```

7584 \pgfrememberpicturepositiononpagetrue
7585 \pgf@relevantforpicturesizefalse
7586 \CT@arc@
7587 \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
7588 \clist_if_in:NnT \l_@@_borders_clist { right }
7589 { \@@_stroke_vertical:n \l_tmpb_tl }
7590 \clist_if_in:NnT \l_@@_borders_clist { left }
7591 { \@@_stroke_vertical:n \l_@@_tmpd_tl }
7592 \clist_if_in:NnT \l_@@_borders_clist { bottom }
7593 { \@@_stroke_horizontal:n \l_tmpa_tl }
7594 \clist_if_in:NnT \l_@@_borders_clist { top }
7595 { \@@_stroke_horizontal:n \l_@@_tmpc_tl }
7596 }
7597 \keys_define:nn { NiceMatrix / OnlyForTikzInBorders }
7598 {
7599   tikz .code:n =
7600     \cs_if_exist:NTF \tikzpicture
7601       { \tl_set:Nn \l_@@_borders_tikz_tl { #1 } }
7602       { \@@_error:n { tikz-in~borders~without~tikz } } ,
7603   tikz .value_required:n = true ,
7604   top .code:n = ,
7605   bottom .code:n = ,
7606   left .code:n = ,
7607   right .code:n = ,
7608   unknown .code:n = \@@_error:n { bad~border }
7609 }

```

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).

```

7610 \cs_new_protected:Npn \@@_stroke_vertical:n #1
7611 {
7612   \@@_qpoint:n \l_@@_tmpc_tl
7613   \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
7614   \@@_qpoint:n \l_tmpa_tl
7615   \dim_set:Nn \l_@@_tmpc_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
7616   \@@_qpoint:n { #1 }
7617   \tl_if_empty:NTF \l_@@_borders_tikz_tl
7618   {
7619     \pgfpathmoveto { \pgfpoint \pgf@x \l_tmpb_dim }
7620     \pgfpathlineto { \pgfpoint \pgf@x \l_@@_tmpc_dim }
7621     \pgfusepathqstroke
7622   }
7623   {
7624     \use:x { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
7625     ( \pgf@x , \l_tmpb_dim ) -- ( \pgf@x , \l_@@_tmpc_dim ) ;
7626   }
7627 }

```

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).

```

7628 \cs_new_protected:Npn \@@_stroke_horizontal:n #1
7629 {
7630   \@@_qpoint:n \l_@@_tmpd_tl
7631   \clist_if_in:NnTF \l_@@_borders_clist { left }
7632   { \dim_set:Nn \l_tmpa_dim { \pgf@x - 0.5 \l_@@_line_width_dim } }
7633   { \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \l_@@_line_width_dim } }
7634   \@@_qpoint:n \l_tmpb_tl
7635   \dim_set:Nn \l_tmpb_dim { \pgf@x + 0.5 \l_@@_line_width_dim }
7636   \@@_qpoint:n { #1 }
7637   \tl_if_empty:NTF \l_@@_borders_tikz_tl
7638   {
7639     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }
7640     \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }

```

```

7641     \pgfusepathqstroke
7642   }
7643   {
7644     \use:x { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
7645     ( \l_tmpa_dim , \pgf@y ) -- ( \l_tmpb_dim , \pgf@y ) ;
7646   }
7647 }

```

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

```

7648 \keys_define:nn { NiceMatrix / BlockBorders }
7649 {
7650   borders .clist_set:N = \l_@@_borders_clist ,
7651   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
7652   rounded-corners .default:n = 4 pt ,
7653   line-width .dim_set:N = \l_@@_line_width_dim
7654 }

```

The following command will be used if the key `tikz` has been used for the command `\Block`. The arguments `#1` and `#2` are the coordinates of the first cell and `#3` and `#4` the coordinates of the last cell of the block. `#5` is a comma-separated list of the Tikz keys used with the path. However, among those keys, you have added in `nicematrix` a special key `offset` (an offset for the rectangle of the block). That's why we have to extract that key first.

```

7655 \cs_new_protected:Npn \@@_block_tikz:nnnnn #1 #2 #3 #4 #5
7656 {
7657   \begin { tikzpicture }
7658     \@@_clip_with_rounded_corners:
7659     \clist_map_inline:nn { #5 }
7660     {
7661       \keys_set_known:nnN { NiceMatrix / SpecialOffset } { ##1 } \l_tmpa_tl
7662       \use:x { \exp_not:N \path [ \l_tmpa_tl ] }
7663       (
7664         [
7665           xshift = \dim_use:N \l_@@_offset_dim ,
7666           yshift = - \dim_use:N \l_@@_offset_dim
7667         ]
7668         #1 -| #2
7669       )
7670       rectangle
7671       (
7672         [
7673           xshift = - \dim_use:N \l_@@_offset_dim ,
7674           yshift = \dim_use:N \l_@@_offset_dim
7675         ]
7676         \int_eval:n { #3 + 1 } -| \int_eval:n { #4 + 1 }
7677       ) ;
7678     }
7679   \end { tikzpicture }
7680 }
7681 \cs_generate_variant:Nn \@@_block_tikz:nnnnn { n n n n V }

7682 \keys_define:nn { NiceMatrix / SpecialOffset }
7683 { offset .dim_set:N = \l_@@_offset_dim }

```

28 How to draw the dotted lines transparently

```

7684 \cs_set_protected:Npn \@@_renew_matrix:
7685 {
7686   \RenewDocumentEnvironment { pmatrix } { } {
7687     { \pNiceMatrix }

```



```

7688     { \endpNiceMatrix }
7689 \RenewDocumentEnvironment { vmatrix } { }
7690     { \vNiceMatrix }
7691     { \endvNiceMatrix }
7692 \RenewDocumentEnvironment { Vmatrix } { }
7693     { \VNiceMatrix }
7694     { \endVNiceMatrix }
7695 \RenewDocumentEnvironment { bmatrix } { }
7696     { \bNiceMatrix }
7697     { \endbNiceMatrix }
7698 \RenewDocumentEnvironment { Bmatrix } { }
7699     { \BNiceMatrix }
7700     { \endBNiceMatrix }
7701 }

```

29 Automatic arrays

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

```

7702 \keys_define:nn { NiceMatrix / Auto }
7703 {
7704     columns-type .tl_set:N = \l_@@_columns_type_tl ,
7705     columns-type .value_required:n = true ,
7706     l .meta:n = { columns-type = l } ,
7707     r .meta:n = { columns-type = r } ,
7708     c .meta:n = { columns-type = c } ,
7709     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
7710     delimiters / color .value_required:n = true ,
7711     delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
7712     delimiters / max-width .default:n = true ,
7713     delimiters .code:n = \keys_set:nn { NiceMatrix / delimiters } { #1 } ,
7714     delimiters .value_required:n = true ,
7715     rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
7716     rounded-corners .default:n = 4 pt
7717 }
7718 \NewDocumentCommand \AutoNiceMatrixWithDelims
7719 { m m O { } } > { \SplitArgument { 1 } { - } } m O { } m ! O { } }
7720 { \@@_auto_nice_matrix:nnnnnn { #1 } { #2 } #4 { #6 } { #3 , #5 , #7 } }
7721 \cs_new_protected:Npn \@@_auto_nice_matrix:nnnnnn #1 #2 #3 #4 #5 #6
7722 {

```

The group is for the protection of the keys.

```

7723 \group_begin:
7724 \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`).

```

7725 % \bool_set_false:N \l_@@_preamble_bool
7726 \use:x
7727 {
7728     \exp_not:N \begin { NiceArrayWithDelims } { #1 } { #2 }
7729     { * { #4 } { \exp_not:V \l_@@_columns_type_tl } }
7730     [ \exp_not:V \l_tmpa_tl ]
7731 }
7732 \int_compare:nNnT \l_@@_first_row_int = 0
7733 {
7734     \int_compare:nNnT \l_@@_first_col_int = 0 { & }
7735     \prg_replicate:nn { #4 - 1 } { & }
7736     \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
7737 }
7738 \prg_replicate:nn { #3 }
7739 {
7740     \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`).

```

7741     \prg_replicate:nn { #4 - 1 } { { } #5 & } #5
7742     \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
7743   }
7744   \int_compare:nNnT \l_@@_last_row_int > { -2 }
7745   {
7746     \int_compare:nNnT \l_@@_first_col_int = 0 { & }
7747     \prg_replicate:nn { #4 - 1 } { & }
7748     \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
7749   }
7750   \end { NiceArrayWithDelims }
7751   \group_end:
7752 }

7753 \cs_set_protected:Npn \@@_define_com:nnn #1 #2 #3
7754 {
7755   \cs_set_protected:cpn { #1 AutoNiceMatrix }
7756   {
7757     \bool_gset_true:N \g_@@_delims_bool
7758     \str_gset:Nx \g_@@_name_env_str { #1 AutoNiceMatrix }
7759     \AutoNiceMatrixWithDelims { #2 } { #3 }
7760   }
7761 }

7762 \@@_define_com:nnn p ( )
7763 \@@_define_com:nnn b [ ]
7764 \@@_define_com:nnn v | |
7765 \@@_define_com:nnn V \ | \ |
7766 \@@_define_com:nnn B \{ \}

```

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

```

7767 \NewDocumentCommand \AutoNiceMatrix { 0 { } m 0 { } m ! 0 { } }
7768 {
7769   \group_begin:
7770   \bool_gset_false:N \g_@@_delims_bool
7771   \AutoNiceMatrixWithDelims . . { #2 } { #4 } [ #1 , #3 , #5 ]
7772   \group_end:
7773 }

```

30 The redefinition of the command `\dotfill`

```

7774 \cs_set_eq:NN \@@_old_dotfill \dotfill
7775 \cs_new_protected:Npn \@@_dotfill:
7776 {

```

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}`).

```

7777   \@@_old_dotfill
7778   \tl_gput_right:Nn \g_@@_cell_after_hook_tl \@@_dotfill_i:
7779 }

```

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

```

7780 \cs_new_protected:Npn \@@_dotfill_i:
7781 { \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.

```

7782 \cs_new_protected:Npn \@@_diagbox:nn #1 #2
7783 {
7784   \tl_gput_right:Nx \g_@@_pre_code_after_tl
7785   {
7786     \@@_actually_diagbox:nnnnnn
7787     { \int_use:N \c@iRow }
7788     { \int_use:N \c@jCol }
7789     { \int_use:N \c@iRow }
7790     { \int_use:N \c@jCol }
7791     { \exp_not:n { #1 } }
7792     { \exp_not:n { #2 } }
7793   }

```

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`.

```

7794   \seq_gput_right:Nx \g_@@_pos_of_blocks_seq
7795   {
7796     { \int_use:N \c@iRow }
7797     { \int_use:N \c@jCol }
7798     { \int_use:N \c@iRow }
7799     { \int_use:N \c@jCol }

```

The last argument is for the name of the block.

```

7800     { }
7801   }
7802 }

```

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.

```

7803 \cs_new_protected:Npn \@@_actually_diagbox:nnnnnn #1 #2 #3 #4 #5 #6
7804 {
7805   \pgfpicture
7806   \pgf@relevantforpicturesizefalse
7807   \pgfrememberpicturepositiononpagetrue
7808   \@@_qpoint:n { row - #1 }
7809   \dim_set_eq:NN \l_tmpa_dim \pgf@y
7810   \@@_qpoint:n { col - #2 }
7811   \dim_set_eq:NN \l_tmpb_dim \pgf@x
7812   \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
7813   \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
7814   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7815   \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
7816   \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
7817   \pgfpathlineto { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
7818   {

```

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.

```

7819     \CT@arc@
7820     \pgfsetroundcap
7821     \pgfusepathqstroke
7822   }
7823   \pgfset { inner~sep = 1 pt }
7824   \pgfscope
7825   \pgftransformshift { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
7826   \pgfnode { rectangle } { south-west }

```

```

7827 {
7828   \begin { minipage } { 20 cm }
7829   \@@_math_toggle_token: #5 \@@_math_toggle_token:
7830   \end { minipage }
7831 }
7832 { }
7833 { }
7834 \endpgfscope
7835 \pgftransformshift { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
7836 \pgfnode { rectangle } { north-east }
7837 {
7838   \begin { minipage } { 20 cm }
7839   \raggedleft
7840   \@@_math_toggle_token: #6 \@@_math_toggle_token:
7841   \end { minipage }
7842 }
7843 { }
7844 { }
7845 \endpgfpicture
7846 }

```

32 The keyword `\CodeAfter`

In fact, in this subsection, we define the user command `\CodeAfter` for the case of the “normal syntax”. For the case of “light-syntax”, see the definition of the environment `{@@-light-syntax}` on p. 79.

In the environments of `nicematrix`, `\CodeAfter` will be linked to `\@@_CodeAfter:`. That macro must *not* be protected since it begins with `\omit`.

```

7847 \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 `\`.

```

7848 \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`.

```

7849 \cs_new_protected:Npn \@@_CodeAfter_ii:n #1 \end
7850 {
7851   \tl_gput_right:Nn \g_nicematrix_code_after_tl { #1 }
7852   \@@_CodeAfter_iv:n
7853 }

```

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

```

7854 \cs_new_protected:Npn \@@_CodeAfter_iv:n #1
7855 {

```

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.

```

7856   \str_if_eq:eeTF \@currenvir { #1 }
7857   { \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`.

```

7858   {
7859     \tl_gput_right:Nn \g_nicematrix_code_after_tl { \end { #1 } }
7860     \@@_CodeAfter_ii:n
7861   }
7862 }

```

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_t1` (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.

```

7863 \cs_new_protected:Npn \@@_delimiter:nnn #1 #2 #3
7864 {
7865   \pgfpicture
7866   \pgfrememberpicturepositiononpagetrue
7867   \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.

```

7868   \@@_qpoint:n { row - 1 }
7869   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
7870   \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
7871   \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).

```

7872   \bool_if:nTF { #3 }
7873   { \dim_set_eq:NN \l_tmpa_dim \c_max_dim }
7874   { \dim_set:Nn \l_tmpa_dim { - \c_max_dim } }
7875   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
7876   {
7877     \cs_if_exist:cT
7878     { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
7879     {
7880       \pgfpointanchor
7881       { \@@_env: - ##1 - #2 }
7882       { \bool_if:nTF { #3 } { west } { east } }
7883       \dim_set:Nn \l_tmpa_dim
7884       { \bool_if:nTF { #3 } \dim_min:nn \dim_max:nn \l_tmpa_dim \pgf@x }
7885     }
7886   }

```

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

```

7887   \pgfset { inner~sep = \c_zero_dim }
7888   \dim_zero:N \nulldelimiterspace
7889   \pgftransformshift
7890   {
7891     \pgfpoint
7892     { \l_tmpa_dim }
7893     { ( \l_@@_y_initial_dim + \l_@@_y_final_dim + \arrayrulewidth ) / 2 }
7894   }
7895   \pgfnode
7896   { rectangle }
7897   { \bool_if:nTF { #3 } { east } { west } }
7898   {

```

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

```

7899     \nullfont
7900     \c_math_toggle_token
7901     \@@_color:V \l_@@_delimiters_color_t1
7902     \bool_if:nTF { #3 } { \left #1 } { \left . }

```

```

7903     \vcenter
7904     {
7905         \nullfont
7906         \hrule \@height
7907             \dim_eval:n { \l_@@_y_initial_dim - \l_@@_y_final_dim }
7908             \@depth \c_zero_dim
7909             \@width \c_zero_dim
7910     }
7911     \bool_if:nTF { #3 } { \right . } { \right #1 }
7912     \c_math_toggle_token
7913 }
7914 { }
7915 { }
7916 \endpgfpicture
7917 }

```

34 The command \SubMatrix

```

7918 \keys_define:nn { NiceMatrix / sub-matrix }
7919 {
7920     extra-height .dim_set:N = \l_@@_submatrix_extra_height_dim ,
7921     extra-height .value_required:n = true ,
7922     left-xshift .dim_set:N = \l_@@_submatrix_left_xshift_dim ,
7923     left-xshift .value_required:n = true ,
7924     right-xshift .dim_set:N = \l_@@_submatrix_right_xshift_dim ,
7925     right-xshift .value_required:n = true ,
7926     xshift .meta:n = { left-xshift = #1, right-xshift = #1 } ,
7927     xshift .value_required:n = true ,
7928     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
7929     delimiters / color .value_required:n = true ,
7930     slim .bool_set:N = \l_@@_submatrix_slim_bool ,
7931     slim .default:n = true ,
7932     hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
7933     hlines .default:n = all ,
7934     vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
7935     vlines .default:n = all ,
7936     hvlines .meta:n = { hlines, vlines } ,
7937     hvlines .value_forbidden:n = true
7938 }
7939 \keys_define:nn { NiceMatrix }
7940 {
7941     SubMatrix .inherit:n = NiceMatrix / sub-matrix ,
7942     NiceArray / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
7943     pNiceArray / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
7944     NiceMatrixOptions / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
7945 }

```

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

```

7946 \keys_define:nn { NiceMatrix / SubMatrix }
7947 {
7948     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
7949     delimiters / color .value_required:n = true ,
7950     hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
7951     hlines .default:n = all ,
7952     vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
7953     vlines .default:n = all ,
7954     hvlines .meta:n = { hlines, vlines } ,
7955     hvlines .value_forbidden:n = true ,
7956     name .code:n =

```

```

7957 \tl_if_empty:nTF { #1 }
7958 { \@@_error:n { Invalid-name } }
7959 {
7960   \regex_match:nnTF { \A[A-Za-z][A-Za-z0-9]*\Z } { #1 }
7961   {
7962     \seq_if_in:NnTF \g_@@_submatrix_names_seq { #1 }
7963     { \@@_error:nn { Duplicate-name-for-SubMatrix } { #1 } }
7964     {
7965       \str_set:Nn \l_@@_submatrix_name_str { #1 }
7966       \seq_gput_right:Nn \g_@@_submatrix_names_seq { #1 }
7967     }
7968   }
7969   { \@@_error:n { Invalid-name } }
7970 } ,
7971 name .value_required:n = true ,
7972 rules .code:n = \keys_set:nn { NiceMatrix / rules } { #1 } ,
7973 rules .value_required:n = true ,
7974 code .tl_set:N = \l_@@_code_tl ,
7975 code .value_required:n = true ,
7976 unknown .code:n = \@@_error:n { Unknown-key-for-SubMatrix }
7977 }

7978 \NewDocumentCommand \@@_SubMatrix_in_code_before { m m m m ! 0 { } }
7979 {
7980   \peek_remove_spaces:n
7981   {
7982     \tl_gput_right:Nx \g_@@_pre_code_after_tl
7983     {
7984       \SubMatrix { #1 } { #2 } { #3 } { #4 }
7985       [
7986         delimiters / color = \l_@@_delimiters_color_tl ,
7987         hlines = \l_@@_submatrix_hlines_clist ,
7988         vlines = \l_@@_submatrix_vlines_clist ,
7989         extra-height = \dim_use:N \l_@@_submatrix_extra_height_dim ,
7990         left-xshift = \dim_use:N \l_@@_submatrix_left_xshift_dim ,
7991         right-xshift = \dim_use:N \l_@@_submatrix_right_xshift_dim ,
7992         slim = \bool_to_str:N \l_@@_submatrix_slim_bool ,
7993         #5
7994       ]
7995     }
7996     \@@_SubMatrix_in_code_before_i { #2 } { #3 }
7997   }
7998 }

7999 \NewDocumentCommand \@@_SubMatrix_in_code_before_i
8000 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
8001 { \@@_SubMatrix_in_code_before_i:nnnn #1 #2 }

8002 \cs_new_protected:Npn \@@_SubMatrix_in_code_before_i:nnnn #1 #2 #3 #4
8003 {
8004   \seq_gput_right:Nx \g_@@_submatrix_seq
8005   {
We use \str_if_eq:nnTF because it is fully expandable.
8006   { \str_if_eq:nnTF { #1 } { last } { \int_use:N \c@iRow } { #1 } }
8007   { \str_if_eq:nnTF { #2 } { last } { \int_use:N \c@jCol } { #2 } }
8008   { \str_if_eq:nnTF { #3 } { last } { \int_use:N \c@iRow } { #3 } }
8009   { \str_if_eq:nnTF { #4 } { last } { \int_use:N \c@jCol } { #4 } }
8010 }
8011 }

```

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`.

```

8012 \hook_gput_code:nnn { begindocument } { . }
8013 {
8014   \tl_set:Nn \l_@@_argspec_tl { m m m m 0 { } E { _ ^ } { { } { } } }
8015   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
8016   \exp_args:NNV \NewDocumentCommand \@@_SubMatrix \l_@@_argspec_tl
8017     {
8018       \peek_remove_spaces:n
8019       {
8020         \@@_sub_matrix:nnnnnnn
8021         { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } { #7 }
8022       }
8023     }
8024 }

```

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).

```

8025 \NewDocumentCommand \@@_compute_i_j:nn
8026 { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
8027 { \@@_compute_i_j:nnnn #1 #2 }
8028 \cs_new_protected:Npn \@@_compute_i_j:nnnn #1 #2 #3 #4
8029 {
8030   \tl_set:Nn \l_@@_first_i_tl { #1 }
8031   \tl_set:Nn \l_@@_first_j_tl { #2 }
8032   \tl_set:Nn \l_@@_last_i_tl { #3 }
8033   \tl_set:Nn \l_@@_last_j_tl { #4 }
8034   \tl_if_eq:NnT \l_@@_first_i_tl { last }
8035     { \tl_set:NV \l_@@_first_i_tl \c@iRow }
8036   \tl_if_eq:NnT \l_@@_first_j_tl { last }
8037     { \tl_set:NV \l_@@_first_j_tl \c@jCol }
8038   \tl_if_eq:NnT \l_@@_last_i_tl { last }
8039     { \tl_set:NV \l_@@_last_i_tl \c@iRow }
8040   \tl_if_eq:NnT \l_@@_last_j_tl { last }
8041     { \tl_set:NV \l_@@_last_j_tl \c@jCol }
8042 }
8043 \cs_new_protected:Npn \@@_sub_matrix:nnnnnnn #1 #2 #3 #4 #5 #6 #7
8044 {
8045   \group_begin:

```

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

```

8046 \@@_compute_i_j:nn { #2 } { #3 }
8047 % added 6.19b
8048 \int_compare:nNnT \l_@@_first_i_tl = \l_@@_last_i_tl
8049 { \cs_set:Npn \arraystretch { 1 } }
8050 \bool_lazy_or:nnTF
8051 { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
8052 { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
8053 { \@@_error:nn { Construct~too~large } { \SubMatrix } }
8054 {
8055   \str_clear_new:N \l_@@_submatrix_name_str

```



```

8056 \keys_set:nn { NiceMatrix / SubMatrix } { #5 }
8057 \pgfpicture
8058 \pgfrememberpicturepositiononpagetrue
8059 \pgf@relevantforpicturesizefalse
8060 \pgfset { inner-sep = \c_zero_dim }
8061 \dim_set_eq:Nn \l_@@_x_initial_dim \c_max_dim
8062 \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }

```

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

```

8063 \bool_if:NTF \l_@@_submatrix_slim_bool
8064 { \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl }
8065 { \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int }
8066 {
8067   \cs_if_exist:cT
8068   { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
8069   {
8070     \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
8071     \dim_set:Nn \l_@@_x_initial_dim
8072     { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
8073   }
8074   \cs_if_exist:cT
8075   { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
8076   {
8077     \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
8078     \dim_set:Nn \l_@@_x_final_dim
8079     { \dim_max:nn \l_@@_x_final_dim \pgf@x }
8080   }
8081 }
8082 \dim_compare:nNnTF \l_@@_x_initial_dim = \c_max_dim
8083 { \@@_error:nn { Impossible-delimiter } { left } }
8084 {
8085   \dim_compare:nNnTF \l_@@_x_final_dim = { - \c_max_dim }
8086   { \@@_error:nn { Impossible-delimiter } { right } }
8087   { \@@_sub_matrix_i:nnnn { #1 } { #4 } { #6 } { #7 } }
8088 }
8089 \endpgfpicture
8090 }
8091 \group_end:
8092 }

```

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

```

8093 \cs_new_protected:Npn \@@_sub_matrix_i:nnnn #1 #2 #3 #4
8094 {
8095   \@@_qpoint:n { row - \l_@@_first_i_tl - base }
8096   \dim_set:Nn \l_@@_y_initial_dim
8097   {
8098     \fp_to_dim:n
8099     {
8100       \pgf@y
8101       + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
8102     }
8103   } % modified 6.13c
8104   \@@_qpoint:n { row - \l_@@_last_i_tl - base }
8105   \dim_set:Nn \l_@@_y_final_dim
8106   { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
8107   % modified 6.13c
8108   \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
8109   {
8110     \cs_if_exist:cT
8111     { pgf @ sh @ ns @ \@@_env: - \l_@@_first_i_tl - ##1 }
8112     {
8113       \pgfpointanchor { \@@_env: - \l_@@_first_i_tl - ##1 } { north }
8114       \dim_set:Nn \l_@@_y_initial_dim

```

```

8115         { \dim_max:nn \l_@@_y_initial_dim \pgf@y }
8116     }
8117     \cs_if_exist:cT
8118     { \pgf @ sh @ ns @ \@@_env: - \l_@@_last_i_tl - ##1 }
8119     {
8120         \pgfpointanchor { \@@_env: - \l_@@_last_i_tl - ##1 } { south }
8121         \dim_set:Nn \l_@@_y_final_dim
8122         { \dim_min:nn \l_@@_y_final_dim \pgf@y }
8123     }
8124 }
8125 \dim_set:Nn \l_tmpa_dim
8126 {
8127     \l_@@_y_initial_dim - \l_@@_y_final_dim +
8128     \l_@@_submatrix_extra_height_dim - \arrayrulewidth
8129 }
8130 \dim_zero:N \nulldelimiterspace

```

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

```

8131 \group_begin:
8132 \pgfsetlinewidth { 1.1 \arrayrulewidth }
8133 \@@_set_CT@arc@:V \l_@@_rules_color_tl
8134 \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`.

```

8135 \seq_map_inline:Nn \g_@@_cols_vlism_seq
8136 {
8137     \int_compare:nNnT \l_@@_first_j_tl < { ##1 }
8138     {
8139         \int_compare:nNnT
8140         { ##1 } < { \int_eval:n { \l_@@_last_j_tl + 1 } }
8141         {

```

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

```

8142         \@@_qpoint:n { col - ##1 }
8143         \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
8144         \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
8145         \pgfusepathqstroke
8146     }
8147 }
8148 }

```

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.

```

8149 \tl_if_eq:NnTF \l_@@_submatrix_vlines_clist { all }
8150 { \int_step_inline:nn { \l_@@_last_j_tl - \l_@@_first_j_tl } }
8151 { \clist_map_inline:Nn \l_@@_submatrix_vlines_clist }
8152 {
8153     \bool_lazy_and:nnTF
8154     { \int_compare_p:nNn { ##1 } > 0 }
8155     {
8156         \int_compare_p:nNn
8157         { ##1 } < { \l_@@_last_j_tl - \l_@@_first_j_tl + 1 } }
8158     {
8159         \@@_qpoint:n { col - \int_eval:n { ##1 + \l_@@_first_j_tl } }
8160         \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
8161         \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
8162         \pgfusepathqstroke
8163     }
8164     { \@@_error:nnn { Wrong-line-in-SubMatrix } { vertical } { ##1 } }
8165 }

```

Now, we draw the horizontal rules specified in the key `hlines` of `\SubMatrix`. The last argument of `\int_step_inline:n` or `\clist_map_inline:Nn` is given by curryfication.

```

8166 \tl_if_eq:NnTF \l_@@_submatrix_hlines_clist { all }
8167 { \int_step_inline:n { \l_@@_last_i_tl - \l_@@_first_i_tl } }
8168 { \clist_map_inline:Nn \l_@@_submatrix_hlines_clist }
8169 {
8170   \bool_lazy_and:nnTF
8171   { \int_compare_p:nNn { ##1 } > 0 }
8172   {
8173     \int_compare_p:nNn
8174     { ##1 } < { \l_@@_last_i_tl - \l_@@_first_i_tl + 1 } }
8175   {
8176     \@@_qpoint:n { row - \int_eval:n { ##1 + \l_@@_first_i_tl } }

```

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

```

8177   \group_begin:
We compute in \l_tmpa_dim the x-value of the left end of the rule.
8178   \dim_set:Nn \l_tmpa_dim
8179   { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
8180   \str_case:nn { #1 }
8181   {
8182     ( { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
8183     [ { \dim_sub:Nn \l_tmpa_dim { 0.2 mm } }
8184     \{ { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
8185     }
8186   \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }

```

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

```

8187   \dim_set:Nn \l_tmpb_dim
8188   { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
8189   \str_case:nn { #2 }
8190   {
8191     ) { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
8192     ] { \dim_add:Nn \l_tmpb_dim { 0.2 mm } }
8193     \} { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
8194   }
8195   \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
8196   \pgfusepathqstroke
8197   \group_end:
8198 }
8199 { \@@_error:nnn { Wrong~line~in~SubMatrix } { horizontal } { ##1 } }
8200 }

```

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).

```

8201 \str_if_empty:NF \l_@@_submatrix_name_str
8202 {
8203   \@@_pgf_rect_node:nnnnn \l_@@_submatrix_name_str
8204   \l_@@_x_initial_dim \l_@@_y_initial_dim
8205   \l_@@_x_final_dim \l_@@_y_final_dim
8206 }
8207 \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`.

```

8208 \begin { pgfscope }
8209 \pgftransformshift
8210 {
8211   \pgfpoint
8212   { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
8213   { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
8214 }

```

```

8215 \str_if_empty:NTF \l_@@_submatrix_name_str
8216 { \@@_node_left:nn #1 { } }
8217 { \@@_node_left:nn #1 { \@@_env: - \l_@@_submatrix_name_str - left } }
8218 \end { pgfscope }

```

Now, we deal with the right delimiter.

```

8219 \pgftransformshift
8220 {
8221   \pgfpoint
8222   { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
8223   { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
8224 }
8225 \str_if_empty:NTF \l_@@_submatrix_name_str
8226 { \@@_node_right:nnnn #2 { } { #3 } { #4 } }
8227 {
8228   \@@_node_right:nnnn #2
8229   { \@@_env: - \l_@@_submatrix_name_str - right } { #3 } { #4 }
8230 }
8231 \cs_set_eq:NN \pgfpointanchor \@@_pgfpointanchor:n
8232 \flag_clear_new:n { nicematrix }
8233 \l_@@_code_tl
8234 }

```

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`.

```

8235 \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.

```

8236 \cs_new_protected:Npn \@@_pgfpointanchor:n #1
8237 {
8238   \use:e
8239   { \exp_not:N \@@_old_pgfpointanchor { \@@_pgfpointanchor_i:nn #1 } }
8240 }

```

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 -.

```

8241 \cs_new:Npn \@@_pgfpointanchor_i:nn #1 #2
8242 { #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.

```

8243 \tl_const:Nn \c_@@_integers_alist_tl
8244 {
8245   { 1 } { } { 2 } { } { 3 } { } { 4 } { } { 5 } { }
8246   { 6 } { } { 7 } { } { 8 } { } { 9 } { } { 10 } { }
8247   { 11 } { } { 12 } { } { 13 } { } { 14 } { } { 15 } { }
8248   { 16 } { } { 17 } { } { 18 } { } { 19 } { } { 20 } { }
8249 }

```

```

8250 \cs_new:Npn \@@_pgfpointanchor_ii:w #1-#2\q_stop
8251 {

```

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`.

```

8252 \tl_if_empty:nTF { #2 }
8253 {
8254   \str_case:nVTF { #1 } \c_@@_integers_alist_tl
8255   {
8256     \flag_raise:n { nicematrix }
8257     \int_if_even:nTF { \flag_height:n { nicematrix } }
8258     { \int_eval:n { #1 + \l_@@_first_i_tl - 1 } }
8259     { \int_eval:n { #1 + \l_@@_first_j_tl - 1 } }
8260   }
8261   { #1 }
8262 }

```

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

```

8263 { \@@_pgfpointanchor_iii:w { #1 } #2 }
8264 }

```

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`).

```

8265 \cs_new:Npn \@@_pgfpointanchor_iii:w #1 #2 -
8266 {
8267   \str_case:nnF { #1 }
8268   {
8269     { row } { row - \int_eval:n { #2 + \l_@@_first_i_tl - 1 } }
8270     { col } { col - \int_eval:n { #2 + \l_@@_first_j_tl - 1 } }
8271   }

```

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

```

8272 {
8273   \int_eval:n { #1 + \l_@@_first_i_tl - 1 }
8274   - \int_eval:n { #2 + \l_@@_first_j_tl - 1 }
8275 }
8276 }

```

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`).

```

8277 \cs_new_protected:Npn \@@_node_left:nn #1 #2
8278 {
8279   \pgfnode
8280   { rectangle }
8281   { east }
8282   {
8283     \nullfont
8284     \c_math_toggle_token
8285     \@@_color:V \l_@@_delimiters_color_tl
8286     \left #1
8287     \vcenter
8288     {
8289       \nullfont
8290       \hrule \@height \l_tmpa_dim
8291       \@depth \c_zero_dim
8292       \@width \c_zero_dim
8293     }
8294     \right .
8295     \c_math_toggle_token
8296   }
8297   { #2 }

```

```

8298     { }
8299 }

```

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.

```

8300 \cs_new_protected:Npn \@@_node_right:nnnn #1 #2 #3 #4
8301 {
8302   \pgfnode
8303     { rectangle }
8304     { west }
8305     {
8306       \nullfont
8307       \c_math_toggle_token
8308       \@@_color:V \l_@@_delimiters_color_tl
8309       \left .
8310       \vcenter
8311       {
8312         \nullfont
8313         \hrule \@height \l_tmpa_dim
8314               \@depth \c_zero_dim
8315               \@width \c_zero_dim
8316       }
8317       \right #1
8318       \tl_if_empty:nF { #3 } { _ { \smash { #3 } } }
8319       ^ { \smash { #4 } }
8320       \c_math_toggle_token
8321     }
8322     { #2 }
8323     { }
8324 }

```

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

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

```

8325 \NewDocumentCommand \@@_UnderBrace { 0 { } m m m 0 { } }
8326 {
8327   \peek_remove_spaces:n
8328   { \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { under } }
8329 }
8330 \NewDocumentCommand \@@_OverBrace { 0 { } m m m 0 { } }
8331 {
8332   \peek_remove_spaces:n
8333   { \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { over } }
8334 }
8335 \keys_define:nn { NiceMatrix / Brace }
8336 {
8337   left-shorten .bool_set:N = \l_@@_brace_left_shorten_bool ,
8338   left-shorten .default:n = true ,
8339   right-shorten .bool_set:N = \l_@@_brace_right_shorten_bool ,
8340   shorten .meta:n = { left-shorten , right-shorten } ,
8341   right-shorten .default:n = true ,
8342   yshift .dim_set:N = \l_@@_brace_yshift_dim ,
8343   yshift .value_required:n = true ,
8344   yshift .initial:n = \c_zero_dim ,
8345   color .tl_set:N = \l_tmpa_tl ,
8346   color .value_required:n = true ,

```

```

8347     unknown .code:n = \@@_error:n { Unknown-key-for-Brace }
8348   }

```

#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`.

```

8349 \cs_new_protected:Npn \@@_brace:nnnnn #1 #2 #3 #4 #5
8350 {
8351   \group_begin:

```

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

```

8352   \@@_compute_i_j:nn { #1 } { #2 }
8353   \bool_lazy_or:nnTF
8354     { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
8355     { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
8356   {
8357     \str_if_eq:nnTF { #5 } { under }
8358     { \@@_error:nn { Construct-too-large } { \UnderBrace } }
8359     { \@@_error:nn { Construct-too-large } { \OverBrace } }
8360   }
8361   {
8362     \tl_clear:N \l_tmpa_tl
8363     \keys_set:nn { NiceMatrix / Brace } { #4 }
8364     \tl_if_empty:NF \l_tmpa_tl { \color { \l_tmpa_tl } }
8365     \pgfpicture
8366     \pgfrememberpicturepositiononpagetrue
8367     \pgf@relevantforpicturesizefalse
8368     \bool_if:NT \l_@@_brace_left_shorten_bool
8369     {
8370       \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
8371       \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
8372       {
8373         \cs_if_exist:cT
8374           { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
8375         {
8376           \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
8377           \dim_set:Nn \l_@@_x_initial_dim
8378             { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
8379         }
8380       }
8381     }
8382     \bool_lazy_or:nnT
8383       { \bool_not_p:n \l_@@_brace_left_shorten_bool }
8384       { \dim_compare_p:nNn \l_@@_x_initial_dim = \c_max_dim }
8385     {
8386       \@@_qpoint:n { col - \l_@@_first_j_tl }
8387       \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
8388     }
8389     \bool_if:NT \l_@@_brace_right_shorten_bool
8390     {
8391       \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
8392       \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
8393       {
8394         \cs_if_exist:cT
8395           { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
8396         {
8397           \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
8398           \dim_set:Nn \l_@@_x_final_dim
8399             { \dim_max:nn \l_@@_x_final_dim \pgf@x }
8400         }
8401       }
8402     }
8403     \bool_lazy_or:nnT
8404       { \bool_not_p:n \l_@@_brace_right_shorten_bool }

```

```

8405     { \dim_compare_p:nNn \l_@@_x_final_dim = { - \c_max_dim } }
8406     {
8407         \@@_qpoint:n { col - \int_eval:n { \l_@@_last_j_tl + 1 } }
8408         \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
8409     }
8410     \pgfset { inner~sep = \c_zero_dim }
8411     \str_if_eq:nnTF { #5 } { under }
8412     { \@@_underbrace_i:n { #3 } }
8413     { \@@_overbrace_i:n { #3 } }
8414     \endpgfpicture
8415 }
8416 \group_end:
8417 }

```

The argument is the text to put above the brace.

```

8418 \cs_new_protected:Npn \@@_overbrace_i:n #1
8419 {
8420     \@@_qpoint:n { row - \l_@@_first_i_tl }
8421     \pgftransformshift
8422     {
8423         \pgfpoint
8424         { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
8425         { \pgf@y + \l_@@_brace_yshift_dim - 3 pt }
8426     }
8427     \pgfnode
8428     { rectangle }
8429     { south }
8430     {
8431         \vbox_top:n
8432         {
8433             \group_begin:
8434             \everycr { }
8435             \halign
8436             {
8437                 \hfil ## \hfil \crrc
8438                 \@@_math_toggle_token: #1 \@@_math_toggle_token: \cr
8439                 \noalign { \skip_vertical:n { 3 pt } \nointerlineskip }
8440                 \c_math_toggle_token
8441                 \overbrace
8442                 {
8443                     \hbox_to_wd:nn
8444                     { \l_@@_x_final_dim - \l_@@_x_initial_dim }
8445                     { }
8446                 }
8447                 \c_math_toggle_token
8448                 \cr
8449             }
8450             \group_end:
8451         }
8452     }
8453     { }
8454     { }
8455 }

```

The argument is the text to put under the brace.

```

8456 \cs_new_protected:Npn \@@_underbrace_i:n #1
8457 {
8458     \@@_qpoint:n { row - \int_eval:n { \l_@@_last_i_tl + 1 } }
8459     \pgftransformshift
8460     {
8461         \pgfpoint
8462         { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
8463         { \pgf@y - \l_@@_brace_yshift_dim + 3 pt }

```



```

8464     }
8465     \pgfnode
8466     { rectangle }
8467     { north }
8468     {
8469         \group_begin:
8470         \everycr { }
8471         \vbox:n
8472         {
8473             \halign
8474             {
8475                 \hfil ## \hfil \crrc
8476                 \c_math_toggle_token
8477                 \underbrace
8478                 {
8479                     \hbox_to_wd:nn
8480                     { \l_@@_x_final_dim - \l_@@_x_initial_dim }
8481                     { }
8482                 }
8483                 \c_math_toggle_token
8484                 \cr
8485                 \noalign { \skip_vertical:n { 3 pt } \nointerlineskip }
8486                 \@@_math_toggle_token: #1 \@@_math_toggle_token: \cr
8487             }
8488         }
8489         \group_end:
8490     }
8491     { }
8492     { }
8493 }

```

36 The command TikzEveryCell

```

8494
8495
8496 \bool_new:N \l_@@_not_empty_bool
8497 \bool_new:N \l_@@_empty_bool
8498
8499 \keys_define:nn { NiceMatrix / TikzEveryCell }
8500 {
8501     not-empty .code:n =
8502         \bool_lazy_or:nnTF
8503         \l_@@_in_code_after_bool
8504         \g_@@_recreate_cell_nodes_bool
8505         { \bool_set_true:N \l_@@_not_empty_bool }
8506         { \@@_error:n { detection-of~empty-cells } } ,
8507     not-empty .value_forbidden:n = true ,
8508     empty .code:n =
8509         \bool_lazy_or:nnTF
8510         \l_@@_in_code_after_bool
8511         \g_@@_recreate_cell_nodes_bool
8512         { \bool_set_true:N \l_@@_empty_bool }
8513         { \@@_error:n { detection-of~empty-cells } } ,
8514     empty .value_forbidden:n = true ,
8515     unknown .code:n = \@@_error:n { Unknown~key~for~TikzEveryCell }
8516 }
8517
8518
8519 \NewDocumentCommand { \@@_TikzEveryCell } { 0 { } m }
8520 {

```

```

8521 \IfPackageLoadedTF { tikz }
8522 {
8523   \group_begin:
8524   \keys_set:nn { NiceMatrix / TikzEveryCell } { #1 }
8525   \tl_set:Nn \l_tmpa_tl { #2 }
8526   \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
8527   { \@@_for_a_block:nnnnn ##1 }
8528   \@@_all_the_cells:
8529   \group_end:
8530 }
8531 { \@@_error:n { TikzEveryCell~without~tikz } }
8532 }
8533
8534 \tl_new:N \@@_i_tl
8535 \tl_new:N \@@_j_tl
8536
8537 \cs_new_protected:Nn \@@_all_the_cells:
8538 {
8539   \int_step_variable:nNn { \int_use:c { c@iRow } } \@@_i_tl
8540   {
8541     \int_step_variable:nNn { \int_use:c { c@jCol } } \@@_j_tl
8542     {
8543       \cs_if_exist:cF { cell - \@@_i_tl - \@@_j_tl }
8544       {
8545         \exp_args:NNx \seq_if_in:NnF \l_@@_corners_cells_seq
8546         { \@@_i_tl - \@@_j_tl }
8547         {
8548           \bool_set_false:N \l_tmpa_bool
8549           \cs_if_exist:cTF
8550           { pgf @ sh @ ns @ \@@_env: - \@@_i_tl - \@@_j_tl }
8551           {
8552             \bool_if:NF \l_@@_empty_bool
8553             { \bool_set_true:N \l_tmpa_bool }
8554           }
8555           {
8556             \bool_if:NF \l_@@_not_empty_bool
8557             { \bool_set_true:N \l_tmpa_bool }
8558           }
8559           \bool_if:NT \l_tmpa_bool
8560           {
8561             \@@_block_tikz:nnnnV
8562             \@@_i_tl \@@_j_tl \@@_i_tl \@@_j_tl \l_tmpa_tl
8563           }
8564         }
8565       }
8566     }
8567   }
8568 }
8569
8570 \cs_new_protected:Nn \@@_for_a_block:nnnnn
8571 {
8572   \bool_if:NF \l_@@_empty_bool
8573   {
8574     \@@_block_tikz:nnnnV
8575     { #1 } { #2 } { #3 } { #4 } \l_tmpa_tl
8576   }
8577   \@@_mark_cells_of_block:nnnn { #1 } { #2 } { #3 } { #4 }
8578 }
8579
8580 \cs_new_protected:Nn \@@_mark_cells_of_block:nnnn
8581 {
8582   \int_step_inline:nnn { #1 } { #3 }
8583   {

```

```

8584     \int_step_inline:nnn { #2 } { #4 }
8585     { \cs_set:cpn { cell - ##1 - ####1 } { } }
8586   }
8587 }
8588
8589

```

37 The command \ShowCellNames

```

8590 \NewDocumentCommand \@@_ShowCellNames_CodeBefore { }
8591 {
8592   \dim_zero_new:N \g_@@_tmpc_dim
8593   \dim_zero_new:N \g_@@_tmpd_dim
8594   \dim_zero_new:N \g_@@_tmpe_dim
8595   \int_step_inline:nn \c@iRow
8596   {
8597     \begin { pgfpicture }
8598     \@@_qpoint:n { row - ##1 }
8599     \dim_set_eq:NN \l_tmpa_dim \pgf@y
8600     \@@_qpoint:n { row - \int_eval:n { ##1 + 1 } }
8601     \dim_gset:Nn \g_tmpa_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
8602     \dim_gset:Nn \g_tmpb_dim { \l_tmpa_dim - \pgf@y }
8603     \bool_if:NTF \l_@@_in_code_after_bool
8604     \end { pgfpicture }
8605     \int_step_inline:nn \c@jCol
8606     {
8607       \hbox_set:Nn \l_tmpa_box
8608       { \normalfont \Large \color { red ! 50 } ##1 - ####1 }
8609       \begin { pgfpicture }
8610       \@@_qpoint:n { col - ####1 }
8611       \dim_gset_eq:NN \g_@@_tmpc_dim \pgf@x
8612       \@@_qpoint:n { col - \int_eval:n { ####1 + 1 } }
8613       \dim_gset:Nn \g_@@_tmpd_dim { \pgf@x - \g_@@_tmpc_dim }
8614       \dim_gset_eq:NN \g_@@_tmpe_dim \pgf@x
8615       \endpgfpicture
8616       \end { pgfpicture }
8617       \fp_set:Nn \l_tmpa_fp
8618       {
8619         \fp_min:nn
8620         {
8621           \fp_min:nn
8622           { \dim_ratio:nn { \g_@@_tmpd_dim } { \box_wd:N \l_tmpa_box } }
8623           { \dim_ratio:nn { \g_tmpb_dim } { \box_ht_plus_dp:N \l_tmpa_box } }
8624         }
8625         { 1.0 }
8626       }
8627       \box_scale:Nnn \l_tmpa_box { \fp_use:N \l_tmpa_fp } { \fp_use:N \l_tmpa_fp }
8628       \pgfpicture
8629       \pgfrememberpicturepositiononpagetrue
8630       \pgf@relevantforpicturesizefalse
8631       \pgftransformshift
8632       {
8633         \pgfpoint
8634         { 0.5 * ( \g_@@_tmpc_dim + \g_@@_tmpe_dim ) }
8635         { \dim_use:N \g_tmpa_dim }
8636       }
8637       \pgfnode
8638       { rectangle }
8639       { center }
8640       { \box_use:N \l_tmpa_box }
8641       { }
8642       { }

```

```

8643         \endpgfpicture
8644     }
8645 }
8646 }
8647 \NewDocumentCommand \@@_ShowCellNames { }
8648 {
8649     \bool_if:NT \l_@@_in_code_after_bool
8650     {
8651         \pgfpicture
8652         \pgfrememberpicturepositiononpagetrue
8653         \pgf@relevantforpicturesizefalse
8654         \pgfpathrectanglecorners
8655             { \@@_qpoint:n { 1 } }
8656             {
8657                 \@@_qpoint:n
8658                 { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
8659             }
8660         \pgfsetfillopacity { 0.75 }
8661         \pgfsetfillcolor { white }
8662         \pgfusepathqfill
8663         \endpgfpicture
8664     }
8665     \dim_zero_new:N \g_@@_tmpc_dim
8666     \dim_zero_new:N \g_@@_tmpd_dim
8667     \dim_zero_new:N \g_@@_tmpe_dim
8668     \int_step_inline:nn \c@iRow
8669     {
8670         \bool_if:NTF \l_@@_in_code_after_bool
8671         {
8672             \pgfpicture
8673             \pgfrememberpicturepositiononpagetrue
8674             \pgf@relevantforpicturesizefalse
8675         }
8676         { \begin { pgfpicture } }
8677         \@@_qpoint:n { row - ##1 }
8678         \dim_set_eq:NN \l_tmpa_dim \pgf@y
8679         \@@_qpoint:n { row - \int_eval:n { ##1 + 1 } }
8680         \dim_gset:Nn \g_tmpa_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
8681         \dim_gset:Nn \g_tmpb_dim { \l_tmpa_dim - \pgf@y }
8682         \bool_if:NTF \l_@@_in_code_after_bool
8683         { \endpgfpicture }
8684         { \end { pgfpicture } }
8685         \int_step_inline:nn \c@jCol
8686         {
8687             \hbox_set:Nn \l_tmpa_box
8688             {
8689                 \normalfont \Large \sffamily \bfseries
8690                 \bool_if:NTF \l_@@_in_code_after_bool
8691                 { \color { red } }
8692                 { \color { red ! 50 } }
8693                 ##1 - ####1
8694             }
8695             \bool_if:NTF \l_@@_in_code_after_bool
8696             {
8697                 \pgfpicture
8698                 \pgfrememberpicturepositiononpagetrue
8699                 \pgf@relevantforpicturesizefalse
8700             }
8701             { \begin { pgfpicture } }
8702             \@@_qpoint:n { col - ####1 }
8703             \dim_gset_eq:NN \g_@@_tmpc_dim \pgf@x
8704             \@@_qpoint:n { col - \int_eval:n { ####1 + 1 } }
8705             \dim_gset:Nn \g_@@_tmpd_dim { \pgf@x - \g_@@_tmpc_dim }

```

```

8706 \dim_gset_eq:NN \g_@@_tmpe_dim \pgf@x
8707 \bool_if:NTF \l_@@_in_code_after_bool
8708 { \endpgfpicture }
8709 { \end { pgfpicture } }
8710 \fp_set:Nn \l_tmpa_fp
8711 {
8712   \fp_min:nn
8713   {
8714     \fp_min:nn
8715     { \dim_ratio:nn { \g_@@_tmpd_dim } { \box_wd:N \l_tmpa_box } }
8716     { \dim_ratio:nn { \g_tmpb_dim } { \box_ht_plus_dp:N \l_tmpa_box } }
8717   }
8718   { 1.0 }
8719 }
8720 \box_scale:Nnn \l_tmpa_box { \fp_use:N \l_tmpa_fp } { \fp_use:N \l_tmpa_fp }
8721 \pgfpicture
8722 \pgfrememberpicturepositiononpagetrue
8723 \pgf@relevantforpicturesizefalse
8724 \pgftransformshift
8725 {
8726   \pgfpoint
8727   { 0.5 * ( \g_@@_tmpc_dim + \g_@@_tmpe_dim ) }
8728   { \dim_use:N \g_tmpa_dim }
8729 }
8730 \pgfnode
8731 { rectangle }
8732 { center }
8733 { \box_use:N \l_tmpa_box }
8734 { }
8735 { }
8736 \endpgfpicture
8737 }
8738 }
8739 }

```

38 We process the options at package loading

We process the options when the package is loaded (with `\usepackage`) but we recommend to use `\NiceMatrixOptions` instead.

We must process these options after the definition of the environment `{NiceMatrix}` because the option `renew-matrix` executes the code `\cs_set_eq:NN \env@matrix \NiceMatrix`.

Of course, the command `\NiceMatrix` must be defined before such an instruction is executed.

The boolean `\g_@@_footnotehyper_bool` will indicate if the option `footnotehyper` is used.

```

8740 \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.

```

8741 \bool_new:N \g_@@_footnote_bool
8742 \msg_new:nnnn { nicematrix } { Unknown-key-for-package }
8743 {
8744   The~key~'\l_keys_key_str'~is-unknown. \\
8745   That~key~will~be~ignored. \\
8746   For~a~list~of~the~available~keys,~type~H~<return>.
8747 }
8748 {
8749   The~available~keys~are~(in~alphabetic~order):~
8750   footnote,~
8751   footnotehyper,~
8752   messages-for-Overleaf,~
8753   no-test-for-array,~

```

```

8754     renew-dots,~and~
8755     renew-matrix.
8756 }

8757 \keys_define:nn { NiceMatrix / Package }
8758 {
8759     renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
8760     renew-dots .value_forbidden:n = true ,
8761     renew-matrix .code:n = \@@_renew_matrix: ,
8762     renew-matrix .value_forbidden:n = true ,
8763     messages-for-Overleaf .bool_set:N = \g_@@_messages_for_Overleaf_bool ,
8764     footnote .bool_set:N = \g_@@_footnote_bool ,
8765     footnotehyper .bool_set:N = \g_@@_footnotehyper_bool ,
8766     no-test-for-array .bool_set:N = \g_@@_no_test_for_array_bool ,
8767     no-test-for-array .default:n = true ,
8768     unknown .code:n = \@@_error:n { Unknown~key~for~package }
8769 }
8770 \ProcessKeysOptions { NiceMatrix / Package }

8771 \@@_msg_new:nn { footnote~with~footnotehyper~package }
8772 {
8773     You~can't~use~the~option~'footnote'~because~the~package~
8774     footnotehyper~has~already~been~loaded.~
8775     If~you~want,~you~can~use~the~option~'footnotehyper'~and~the~footnotes~
8776     within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
8777     of~the~package~footnotehyper.\\
8778     The~package~footnote~won't~be~loaded.
8779 }

8780 \@@_msg_new:nn { footnotehyper~with~footnote~package }
8781 {
8782     You~can't~use~the~option~'footnotehyper'~because~the~package~
8783     footnote~has~already~been~loaded.~
8784     If~you~want,~you~can~use~the~option~'footnote'~and~the~footnotes~
8785     within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
8786     of~the~package~footnote.\\
8787     The~package~footnotehyper~won't~be~loaded.
8788 }

8789 \bool_if:NT \g_@@_footnote_bool
8790 {

```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```

8791     \IfClassLoadedTF { beamer }
8792     { \bool_set_false:N \g_@@_footnote_bool }
8793     {
8794         \IfPackageLoadedTF { footnotehyper }
8795         { \@@_error:n { footnote~with~footnotehyper~package } }
8796         { \usepackage { footnote } }
8797     }
8798 }

8799 \bool_if:NT \g_@@_footnotehyper_bool
8800 {

```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```

8801     \IfClassLoadedTF { beamer }
8802     { \bool_set_false:N \g_@@_footnote_bool }
8803     {
8804         \IfPackageLoadedTF { footnote }
8805         { \@@_error:n { footnotehyper~with~footnote~package } }
8806         { \usepackage { footnotehyper } }

```

```

8807     }
8808     \bool_set_true:N \g_@@_footnote_bool
8809 }

```

The flag `\g_@@_footnote_bool` is raised and so, we will only have to test `\g_@@_footnote_bool` in order to know if we have to insert an environment `{savenotes}`.

39 About the package underscore

If the user loads the package `underscore`, it must be loaded *before* the package `nicematrix`. If it is loaded after, we raise an error.

```

8810 \bool_new:N \l_@@_underscore_loaded_bool
8811 \IfPackageLoadedTF { underscore }
8812 { \bool_set_true:N \l_@@_underscore_loaded_bool }
8813 { }
8814 \hook_gput_code:nnn { begindocument } { . }
8815 {
8816     \bool_if:NF \l_@@_underscore_loaded_bool
8817     {
8818         \IfPackageLoadedTF { underscore }
8819         { \@@_error:n { underscore~after~nicematrix } }
8820         { }
8821     }
8822 }

```

40 Error messages of the package

```

8823 \bool_if:NTF \g_@@_messages_for_Overleaf_bool
8824 { \str_const:Nn \c_@@_available_keys_str { } }
8825 {
8826     \str_const:Nn \c_@@_available_keys_str
8827     { For~a~list~of~the~available~keys,~type~H~<return>. }
8828 }
8829 \seq_new:N \g_@@_types_of_matrix_seq
8830 \seq_gset_from_clist:Nn \g_@@_types_of_matrix_seq
8831 {
8832     NiceMatrix ,
8833     pNiceMatrix , bNiceMatrix , vNiceMatrix, BNiceMatrix, VNiceMatrix
8834 }
8835 \seq_gset_map_x:NNn \g_@@_types_of_matrix_seq \g_@@_types_of_matrix_seq
8836 { \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`.

```

8837 \cs_new_protected:Npn \@@_error_too_much_cols:
8838 {
8839     \seq_if_in:NVTF \g_@@_types_of_matrix_seq \g_@@_name_env_str
8840     {
8841         \int_compare:nNnTF \l_@@_last_col_int = { -2 }
8842         { \@@_fatal:n { too-much-cols-for-matrix } }
8843         {

```

```

8844         \int_compare:nNnTF \l_@@_last_col_int = { -1 }
8845         { \@@_fatal:n { too-much-cols-for-matrix } }
8846         {
8847             \bool_if:NF \l_@@_last_col_without_value_bool
8848             { \@@_fatal:n { too-much-cols-for-matrix-with-last-col } }
8849         }
8850     }
8851 }
8852 { \@@_fatal:nn { too-much-cols-for-array } }
8853 }

```

The following command must *not* be protected since it's used in an error message.

```

8854 \cs_new:Npn \@@_message_hdotsfor:
8855 {
8856     \tl_if_empty:VF \g_@@_HVdotsfor_lines_tl
8857     { ~Maybe-your-use-of~\token_to_str:N \Hdotsfor\ is-incorrect.}
8858 }
8859 \@@_msg_new:nn { hvlines,~rounded-corners-and~corners }
8860 {
8861     Incompatible~options.\\
8862     You-should-not-use~'hvlines',~'rounded-corners'~and~'corners'~at~this~time.\\
8863     The~output~will~not~be~reliable.
8864 }
8865 \@@_msg_new:nn { negative-weight }
8866 {
8867     Negative~weight.\\
8868     The~weight~of~the~'X'~columns~must~be~positive~and~you~have~used~
8869     the~value~'\int_use:N \l_@@_weight_int'.\\
8870     The~absolute~value~will~be~used.
8871 }
8872 \@@_msg_new:nn { last-col-not-used }
8873 {
8874     Column~not~used.\\
8875     The~key~'last-col'~is~in~force~but~you~have~not~used~that~last~column~
8876     in~your~\@@_full_name_env:~.~However,~you~can~go~on.
8877 }
8878 \@@_msg_new:nn { too-much-cols-for-matrix-with-last-col }
8879 {
8880     Too-much-columns.\\
8881     In~the~row~\int_eval:n { \c@iRow },~
8882     you~try~to~use~more~columns~
8883     than~allowed~by~your~\@@_full_name_env:~.\@@_message_hdotsfor:\\
8884     The~maximal~number~of~columns~is~\int_eval:n { \l_@@_last_col_int - 1 }~
8885     (plus~the~exterior~columns).~This~error~is~fatal.
8886 }
8887 \@@_msg_new:nn { too-much-cols-for-matrix }
8888 {
8889     Too-much-columns.\\
8890     In~the~row~\int_eval:n { \c@iRow },~
8891     you~try~to~use~more~columns~than~allowed~by~your~
8892     \@@_full_name_env:~.\@@_message_hdotsfor:\\ Recall~that~the~maximal~
8893     number~of~columns~for~a~matrix~(excepted~the~potential~exterior~
8894     columns)~is~fixed~by~the~LaTeX~counter~'MaxMatrixCols'.~
8895     Its~current~value~is~\int_use:N \c@MaxMatrixCols\ (use~
8896     \token_to_str:N \setcounter\ to~change~that~value).~
8897     This~error~is~fatal.
8898 }
8899 \@@_msg_new:nn { too-much-cols-for-array }
8900 {
8901     Too-much-columns.\\
8902     In~the~row~\int_eval:n { \c@iRow },~

```



```

8903 ~you-try-to-use-more-columns-than-allowed-by-your~
8904 \@@_full_name_env:.\@@_message_hdotsfor:\ The-maximal-number-of-columns-is~
8905 \int_use:N \g_@@_static_num_of_col_int\
8906 ~(plus-the-potential-exterior-ones).
8907 This-error-is-fatal.
8908 }
8909 \@@_msg_new:nn { columns-not-used }
8910 {
8911 Columns-not-used.\\
8912 The-preamble-of-your~\@@_full_name_env:\ announces~\int_use:N
8913 \g_@@_static_num_of_col_int\ columns-but-you-use-only~\int_use:N \c@jCol.\\
8914 The-columns-you-did-not-used-won't-be-created.\\
8915 You-won't-have-similar-error-till-the-end-of-the-document.
8916 }
8917 \@@_msg_new:nn { in-first-col }
8918 {
8919 Erroneous-use.\\
8920 You-can't-use-the-command~#1 in-the-first-column-(number~0)-of-the-array.\\
8921 That-command-will-be-ignored.
8922 }
8923 \@@_msg_new:nn { in-last-col }
8924 {
8925 Erroneous-use.\\
8926 You-can't-use-the-command~#1 in-the-last-column-(exterior)-of-the-array.\\
8927 That-command-will-be-ignored.
8928 }
8929 \@@_msg_new:nn { in-first-row }
8930 {
8931 Erroneous-use.\\
8932 You-can't-use-the-command~#1 in-the-first-row-(number~0)-of-the-array.\\
8933 That-command-will-be-ignored.
8934 }
8935 \@@_msg_new:nn { in-last-row }
8936 {
8937 You-can't-use-the-command~#1 in-the-last-row-(exterior)-of-the-array.\\
8938 That-command-will-be-ignored.
8939 }
8940 \@@_msg_new:nn { caption-outside-float }
8941 {
8942 Key-caption-forbidden.\\
8943 You-can't-use-the-key~'caption'~because-you-are-not-in-a-floating~
8944 environment.~This-key-will-be-ignored.
8945 }
8946 \@@_msg_new:nn { short-caption-without-caption }
8947 {
8948 You-should-not-use-the-key~'short-caption'~without~'caption'.~
8949 However,~your~'short-caption'~will-be-used-as~'caption'.
8950 }
8951 \@@_msg_new:nn { double-closing-delimiter }
8952 {
8953 Double-delimiter.\\
8954 You-can't-put-a-second-closing-delimiter~"#1"~just-after-a-first-closing~
8955 delimiter.~This-delimiter-will-be-ignored.
8956 }
8957 \@@_msg_new:nn { delimiter-after-opening }
8958 {
8959 Double-delimiter.\\
8960 You-can't-put-a-second-delimiter~"#1"~just-after-a-first-opening~
8961 delimiter.~That-delimiter-will-be-ignored.
8962 }

```

```

8963 \@@_msg_new:nn { bad-option-for-line-style }
8964 {
8965   Bad~line~style.\\
8966   Since~you~haven't~loaded~Tikz,~the~only~value~you~can~give~to~'line-style'~
8967   is~'standard'.~That~key~will~be~ignored.
8968 }
8969 \@@_msg_new:nn { Identical~notes~in~caption }
8970 {
8971   Identical~tabular~notes.\\
8972   You~can't~put~several~notes~with~the~same~content~in~
8973   \token_to_str:N \caption\ (but~you~can~in~the~main~tabular).\\
8974   If~you~go~on,~the~output~will~probably~be~erroneous.
8975 }
8976 \@@_msg_new:nn { tabularnote~below~the~tabular }
8977 {
8978   \token_to_str:N \tabularnote\ forbidden\\
8979   You~can't~use~\token_to_str:N \tabularnote\ in~the~caption~
8980   of~your~tabular~because~the~caption~will~be~composed~below~
8981   the~tabular.~If~you~want~the~caption~above~the~tabular~use~the~
8982   key~'caption~above'~in~\token_to_str:N \NiceMatrixOptions.\\
8983   Your~\token_to_str:N \tabularnote\ will~be~discarded~and~
8984   no~similar~error~will~raised~in~this~document.
8985 }
8986 \@@_msg_new:nn { Unknown~key~for~rules }
8987 {
8988   Unknown~key.\\
8989   There~is~only~two~keys~available~here:~width~and~color.\\
8990   Your~key~'\l_keys_key_str'~will~be~ignored.
8991 }
8992 \@@_msg_new:nn { Unknown~key~for~TikzEveryCell }
8993 {
8994   Unknown~key.\\
8995   There~is~only~two~keys~available~here:~
8996   'empty'~and~'not-empty'.\\
8997   Your~key~'\l_keys_key_str'~will~be~ignored.
8998 }
8999 \@@_msg_new:nn { Unknown~key~for~rotate }
9000 {
9001   Unknown~key.\\
9002   The~only~key~available~here~is~'c'.\\
9003   Your~key~'\l_keys_key_str'~will~be~ignored.
9004 }
9005 \@@_msg_new:nnn { Unknown~key~for~custom~line }
9006 {
9007   Unknown~key.\\
9008   The~key~'\l_keys_key_str'~is~unknown~in~a~'custom~line'.~
9009   It~you~go~on,~you~will~probably~have~other~errors. \\
9010   \c_@@_available_keys_str
9011 }
9012 {
9013   The~available~keys~are~(in~alphabetic~order):~
9014   ccommand,~
9015   color,~
9016   command,~
9017   dotted,~
9018   letter,~
9019   multiplicity,~
9020   sep-color,~
9021   tikz,~and~total-width.
9022 }
9023 \@@_msg_new:nnn { Unknown~key~for~xdots }

```

```

9024 {
9025   Unknown~key.\\
9026   The~key~'\l_keys_key_str'~is~unknown~for~a~command~for~drawing~dotted~rules.\\
9027   \c_@@_available_keys_str
9028 }
9029 {
9030   The~available~keys~are~(in~alphabetic~order):~
9031   'color',~
9032   'horizontal-labels',~
9033   'inter',~
9034   'line-style',~
9035   'radius',~
9036   'shorten',~
9037   'shorten-end'~and~'shorten-start'.
9038 }
9039 \@@_msg_new:nn { Unknown~key~for~rowcolors }
9040 {
9041   Unknown~key.\\
9042   As~for~now,~there~is~only~two~keys~available~here:~'cols'~and~'respect-blocks'~
9043   (and~you~try~to~use~'\l_keys_key_str')\\
9044   That~key~will~be~ignored.
9045 }
9046 \@@_msg_new:nn { label~without~caption }
9047 {
9048   You~can't~use~the~key~'label'~in~your~'{NiceTabular}'~because~
9049   you~have~not~used~the~key~'caption'.~The~key~'label'~will~be~ignored.
9050 }
9051 \@@_msg_new:nn { W~warning }
9052 {
9053   Line~\msg_line_number:~.~The~cell~is~too~wide~for~your~column~'W'~
9054   (row~\int_use:N \c@iRow).
9055 }
9056 \@@_msg_new:nn { Construct~too~large }
9057 {
9058   Construct~too~large.\\
9059   Your~command~\token_to_str:N #1
9060   can't~be~drawn~because~your~matrix~is~too~small.\\
9061   That~command~will~be~ignored.
9062 }
9063 \@@_msg_new:nn { underscore~after~nicematrix }
9064 {
9065   Problem~with~'underscore'.\\
9066   The~package~'underscore'~should~be~loaded~before~'nicematrix'.~
9067   You~can~go~on~but~you~won't~be~able~to~write~something~such~as:\\
9068   '\token_to_str:N \Cdots\token_to_str:N _{n~\token_to_str:N \text{~times}}'.
9069 }
9070 \@@_msg_new:nn { ampersand~in~light-syntax }
9071 {
9072   Ampersand~forbidden.\\
9073   You~can't~use~an~ampersand~(\token_to_str:N &)~to~separate~columns~because~
9074   the~key~'light-syntax'~is~in~force.~This~error~is~fatal.
9075 }
9076 \@@_msg_new:nn { double~backslash~in~light-syntax }
9077 {
9078   Double~backslash~forbidden.\\
9079   You~can't~use~\token_to_str:N
9080   \\~to~separate~rows~because~the~key~'light-syntax'~
9081   is~in~force.~You~must~use~the~character~'\l_@@_end_of_row_tl'~
9082   (set~by~the~key~'end-of-row').~This~error~is~fatal.
9083 }

```

```

9084 \@@_msg_new:nn { hlines-with-color }
9085 {
9086   Incompatible~keys.\\
9087   You~can't~use~the~keys~'hlines',~'vlines'~or~'hvlines'~for~a~
9088   '\token_to_str:N \Block'~when~the~key~'color'~or~'draw'~is~used.\\
9089   Maybe~it~will~possible~in~future~version.\\
9090   Your~key~will~be~discarded.
9091 }
9092 \@@_msg_new:nn { bad-value-for-baseline }
9093 {
9094   Bad~value~for~baseline.\\
9095   The~value~given~to~'baseline'~(\int_use:N \l_tmpa_int)~is~not~
9096   valid.~The~value~must~be~between~\int_use:N \l_@@_first_row_int\ and~
9097   \int_use:N \g_@@_row_total_int\ or~equal~to~'t',~'c'~or~'b'~or~of~
9098   the~form~'line-i'.\\
9099   A~value~of~1~will~be~used.
9100 }
9101 \@@_msg_new:nn { detection-of-empty-cells }
9102 {
9103   Problem~with~'not-empty'\\
9104   For~technical~reasons,~you~must~activate~
9105   'recreate-cell-nodes'~in~\token_to_str:N \CodeBefore\
9106   in~order~to~use~the~key~'\l_keys_key_str'.\\
9107   That~key~will~be~ignored.
9108 }
9109 \@@_msg_new:nn { ragged2e-not-loaded }
9110 {
9111   You~have~to~load~'ragged2e'~in~order~to~use~the~key~'\l_keys_key_str'~in~
9112   your~column~'\l_@@_vpos_col_str'~(or~'X').~The~key~'\str_lowercase:V
9113   \l_keys_key_str'~will~be~used~instead.
9114 }
9115 \@@_msg_new:nn { Invalid-name }
9116 {
9117   Invalid~name.\\
9118   You~can't~give~the~name~'\l_keys_value_tl'~to~a~\token_to_str:N
9119   \SubMatrix\ of~your~\@@_full_name_env:~\\
9120   A~name~must~be~accepted~by~the~regular~expression~[A-Za-z][A-Za-z0-9]*.\\
9121   This~key~will~be~ignored.
9122 }
9123 \@@_msg_new:nn { Wrong-line-in-SubMatrix }
9124 {
9125   Wrong~line.\\
9126   You~try~to~draw~a~#1~line~of~number~'#2'~in~a~
9127   \token_to_str:N \SubMatrix\ of~your~\@@_full_name_env:\ but~that~
9128   number~is~not~valid.~It~will~be~ignored.
9129 }
9130 \@@_msg_new:nn { Impossible-delimiter }
9131 {
9132   Impossible~delimiter.\\
9133   It's~impossible~to~draw~the~#1~delimiter~of~your~
9134   \token_to_str:N \SubMatrix\ because~all~the~cells~are~empty~
9135   in~that~column.
9136   \bool_if:NT \l_@@_submatrix_slim_bool
9137   { ~Maybe~you~should~try~without~the~key~'slim'. } \\
9138   This~\token_to_str:N \SubMatrix\ will~be~ignored.
9139 }
9140 \@@_msg_new:nnn { width-without-X-columns }
9141 {
9142   You~have~used~the~key~'width'~but~you~have~put~no~'X'~column.~
9143   That~key~will~be~ignored.
9144 }

```

```

9145 {
9146   This-message-is-the-message~'width-without~X~columns'~
9147   of~the-module~'nicematrix'.~
9148   The~experimented~users~can~disable~that~message~with~
9149   \token_to_str:N \msg_redirect_name:nnn.\\
9150 }
9151
9152 \@@_msg_new:nn { key~multiplicity~with~dotted }
9153 {
9154   Incompatible~keys. \\
9155   You~have~used~the~key~'multiplicity'~with~the~key~'dotted'~
9156   in~a~'custom-line'.~They~are~incompatible. \\
9157   The~key~'multiplicity'~will~be~discarded.
9158 }
9159 \@@_msg_new:nn { empty~environment }
9160 {
9161   Empty~environment.\\
9162   Your~\@@_full_name_env:\ is~empty.~This~error~is~fatal.
9163 }
9164 \@@_msg_new:nn { No~letter~and~no~command }
9165 {
9166   Erroneous~use.\\
9167   Your~use~of~'custom-line'~is~no~op~since~you~don't~have~used~the~
9168   key~'letter'~(for~a~letter~for~vertical~rules)~nor~the~keys~'command'~or~
9169   '~'ccommand'~(to~draw~horizontal~rules).\\
9170   However,~you~can~go~on.
9171 }
9172 \@@_msg_new:nn { Forbidden~letter }
9173 {
9174   Forbidden~letter.\\
9175   You~can't~use~the~letter~'\l_@@_letter_str'~for~a~customized~line.\\
9176   It~will~be~ignored.
9177 }
9178 \@@_msg_new:nn { Several~letters }
9179 {
9180   Wrong~name.\\
9181   You~must~use~only~one~letter~as~value~for~the~key~'letter'~(and~you~
9182   have~used~'\l_@@_letter_str').\\
9183   It~will~be~ignored.
9184 }
9185 \@@_msg_new:nn { Delimiter~with~small }
9186 {
9187   Delimiter~forbidden.\\
9188   You~can't~put~a~delimiter~in~the~preamble~of~your~\@@_full_name_env:\
9189   because~the~key~'small'~is~in~force.\\
9190   This~error~is~fatal.
9191 }
9192 \@@_msg_new:nn { unknown~cell~for~line~in~CodeAfter }
9193 {
9194   Unknown~cell.\\
9195   Your~command~\token_to_str:N\line\{#1\}\{#2\}~in~
9196   the~\token_to_str:N \CodeAfter\ of~your~\@@_full_name_env:\
9197   can't~be~executed~because~a~cell~doesn't~exist.\\
9198   This~command~\token_to_str:N \line\ will~be~ignored.
9199 }
9200 \@@_msg_new:nnn { Duplicate~name~for~SubMatrix }
9201 {
9202   Duplicate~name.\\
9203   The~name~'#1'~is~already~used~for~a~\token_to_str:N \SubMatrix\
9204   in~this~\@@_full_name_env:.\

```

```

9205 This~key~will~be~ignored.\\
9206 \bool_if:NF \g_@@_messages_for_Overleaf_bool
9207 { For~a~list~of~the~names~already~used,~type~H~<return>. }
9208 }
9209 {
9210 The~names~already~defined~in~this~\@@_full_name_env:\ are:~
9211 \seq_use:Nnnn \g_@@_submatrix_names_seq { ~and~ } { ,~ } { ~and~ }.
9212 }
9213 \@@_msg_new:nn { r-or-l-with-preamble }
9214 {
9215 Erroneous~use.\\
9216 You~can't~use~the~key~'\l_keys_key_str'~in~your~\@@_full_name_env:~
9217 You~must~specify~the~alignment~of~your~columns~with~the~preamble~of~
9218 your~\@@_full_name_env:~
9219 This~key~will~be~ignored.
9220 }
9221 \@@_msg_new:nn { Hdotsfor~in~col~0 }
9222 {
9223 Erroneous~use.\\
9224 You~can't~use~\token_to_str:N \Hdotsfor\ in~an~exterior~column~of~
9225 the~array.~This~error~is~fatal.
9226 }
9227 \@@_msg_new:nn { bad~corner }
9228 {
9229 Bad~corner.\\
9230 #1~is~an~incorrect~specification~for~a~corner~(in~the~key~
9231 'corners').~The~available~values~are:~NW,~SW,~NE~and~SE.\\
9232 This~specification~of~corner~will~be~ignored.
9233 }
9234 \@@_msg_new:nn { bad~border }
9235 {
9236 Bad~border.\\
9237 \l_keys_key_str\space~is~an~incorrect~specification~for~a~border~
9238 (in~the~key~'borders'~of~the~command~\token_to_str:N \Block).~
9239 The~available~values~are:~left,~right,~top~and~bottom~(and~you~can~
9240 also~use~the~key~'tikz'
9241 \IfPackageLoadedTF { tikz }
9242 { }
9243 {~if~you~load~the~LaTeX~package~'tikz'}).\\
9244 This~specification~of~border~will~be~ignored.
9245 }
9246 \@@_msg_new:nn { TikzEveryCell~without~tikz }
9247 {
9248 TikZ~not~loaded.\\
9249 You~can't~use~\token_to_str:N \TikzEveryCell\
9250 because~you~have~not~loaded~tikz.~
9251 This~command~will~be~ignored.
9252 }
9253 \@@_msg_new:nn { tikz~key~without~tikz }
9254 {
9255 TikZ~not~loaded.\\
9256 You~can't~use~the~key~'tikz'~for~the~command~'\token_to_str:N
9257 \Block'~because~you~have~not~loaded~tikz.~
9258 This~key~will~be~ignored.
9259 }
9260 \@@_msg_new:nn { last~col~non~empty~for~NiceArray }
9261 {
9262 Erroneous~use.\\
9263 In~the~\@@_full_name_env:,~you~must~use~the~key~
9264 'last~col'~without~value.\\
9265 However,~you~can~go~on~for~this~time~

```

```

9266     (the~value~'\l_keys_value_tl'~will~be~ignored).
9267 }

9268 \@@_msg_new:nn { last-col~non~empty~for~NiceMatrixOptions }
9269 {
9270     Erroneous~use.\\
9271     In~\token_to_str:N \NiceMatrixOptions,~you~must~use~the~key~
9272     'last-col'~without~value.\\
9273     However,~you~can~go~on~for~this~time~
9274     (the~value~'\l_keys_value_tl'~will~be~ignored).
9275 }

9276 \@@_msg_new:nn { Block~too~large~1 }
9277 {
9278     Block~too~large.\\
9279     You~try~to~draw~a~block~in~the~cell~#1-#2~of~your~matrix~but~the~matrix~is~
9280     too~small~for~that~block. \\
9281 }

9282 \@@_msg_new:nn { Block~too~large~2 }
9283 {
9284     Block~too~large.\\
9285     The~preamble~of~your~\@@_full_name_env:\ announces~\int_use:N
9286     \g_@@_static_num_of_col_int\
9287     columns~but~you~use~only~\int_use:N \c@jCol\ and~that's~why~a~block~
9288     specified~in~the~cell~#1-#2~can't~be~drawn.~You~should~add~some~ampersands~
9289     (&)~at~the~end~of~the~first~row~of~your~
9290     \@@_full_name_env:.\\
9291     This~block~and~maybe~others~will~be~ignored.
9292 }

9293 \@@_msg_new:nn { unknown~column~type }
9294 {
9295     Bad~column~type.\\
9296     The~column~type~'#1'~in~your~\@@_full_name_env:\
9297     is~unknown. \\
9298     This~error~is~fatal.
9299 }

9300 \@@_msg_new:nn { unknown~column~type~S }
9301 {
9302     Bad~column~type.\\
9303     The~column~type~'S'~in~your~\@@_full_name_env:\ is~unknown. \\
9304     If~you~want~to~use~the~column~type~'S'~of~siunitx,~you~should~
9305     load~that~package. \\
9306     This~error~is~fatal.
9307 }

9308 \@@_msg_new:nn { tabularnote~forbidden }
9309 {
9310     Forbidden~command.\\
9311     You~can't~use~the~command~\token_to_str:N\ tabularnote\
9312     ~here.~This~command~is~available~only~in~
9313     \{NiceTabular\},~\{NiceTabular*\}~and~\{NiceTabularX\}~or~in~
9314     the~argument~of~a~command~\token_to_str:N \caption\ included~
9315     in~an~environment~{table}. \\
9316     This~command~will~be~ignored.
9317 }

9318 \@@_msg_new:nn { borders~forbidden }
9319 {
9320     Forbidden~key.\\
9321     You~can't~use~the~key~'borders'~of~the~command~\token_to_str:N \Block\
9322     because~the~option~'rounded~corners'~
9323     is~in~force~with~a~non~zero~value.\\
9324     This~key~will~be~ignored.
9325 }

```

```

9326 \@@_msg_new:nn { bottomrule-without-booktabs }
9327 {
9328     booktabs~not~loaded.\\
9329     You~can't~use~the~key~'tabular/bottomrule'~because~you~haven't~
9330     loaded~'booktabs'.\\
9331     This~key~will~be~ignored.
9332 }
9333 \@@_msg_new:nn { enumitem~not~loaded }
9334 {
9335     enumitem~not~loaded.\\
9336     You~can't~use~the~command~\token_to_str:N\tabularnote\
9337     ~because~you~haven't~loaded~'enumitem'.\\
9338     All~the~commands~\token_to_str:N\tabularnote\ will~be~
9339     ignored~in~the~document.
9340 }
9341 \@@_msg_new:nn { tikz~without~tikz }
9342 {
9343     Tikz~not~loaded.\\
9344     You~can't~use~the~key~'tikz'~here~because~Tikz~is~not~
9345     loaded.~If~you~go~on,~that~key~will~be~ignored.
9346 }
9347 \@@_msg_new:nn { tikz~in~custom~line~without~tikz }
9348 {
9349     Tikz~not~loaded.\\
9350     You~have~used~the~key~'tikz'~in~the~definition~of~a~
9351     customized~line~(with~'custom~line')~but~tikz~is~not~loaded.~
9352     You~can~go~on~but~you~will~have~another~error~if~you~actually~
9353     use~that~custom~line.
9354 }
9355 \@@_msg_new:nn { tikz~in~borders~without~tikz }
9356 {
9357     Tikz~not~loaded.\\
9358     You~have~used~the~key~'tikz'~in~a~key~'borders'~(of~a~
9359     command~\token_to_str:N\Block')~but~tikz~is~not~loaded.~
9360     That~key~will~be~ignored.
9361 }
9362 \@@_msg_new:nn { color~in~custom~line~with~tikz }
9363 {
9364     Erroneous~use.\\
9365     In~a~'custom~line',~you~have~used~both~'tikz'~and~'color',~
9366     which~is~forbidden~(you~should~use~'color'~inside~the~key~'tikz').~
9367     The~key~'color'~will~be~discarded.
9368 }
9369 \@@_msg_new:nn { Wrong~last~row }
9370 {
9371     Wrong~number.\\
9372     You~have~used~'last~row=\int_use:N \l_@@_last_row_int'~but~your~
9373     \@@_full_name_env:\ seems~to~have~\int_use:N \c@iRow \ rows.~
9374     If~you~go~on,~the~value~of~\int_use:N \c@iRow \ will~be~used~for~
9375     last~row.~You~can~avoid~this~problem~by~using~'last~row'~
9376     without~value~(more~compilations~might~be~necessary).
9377 }
9378 \@@_msg_new:nn { Yet~in~env }
9379 {
9380     Nested~environments.\\
9381     Environments~of~nicematrix~can't~be~nested.\\
9382     This~error~is~fatal.
9383 }
9384 \@@_msg_new:nn { Outside~math~mode }
9385 {

```



```

9386 Outside~math~mode.\\
9387 The~\@@_full_name_env:\ can~be~used~only~in~math~mode~
9388 (and~not~in~\token_to_str:N \vcenter).\\
9389 This~error~is~fatal.
9390 }
9391 \@@_msg_new:nn { One~letter~allowed }
9392 {
9393   Bad~name.\\
9394   The~value~of~key~'\l_keys_key_str'~must~be~of~length~1.\\
9395   It~will~be~ignored.
9396 }
9397 \@@_msg_new:nn { TabularNote~in~CodeAfter }
9398 {
9399   Environment~{TabularNote}~forbidden.\\
9400   You~must~use~{TabularNote}~at~the~end~of~your~{NiceTabular}~
9401   but~*before*~the~\token_to_str:N \CodeAfter.\\
9402   This~environment~{TabularNote}~will~be~ignored.
9403 }
9404 \@@_msg_new:nn { varwidth~not~loaded }
9405 {
9406   varwidth~not~loaded.\\
9407   You~can't~use~the~column~type~'V'~because~'varwidth'~is~not~
9408   loaded.\\
9409   Your~column~will~behave~like~'p'.
9410 }
9411 \@@_msg_new:nnn { Unknow~key~for~RulesBis }
9412 {
9413   Unknow~key.\\
9414   Your~key~'\l_keys_key_str'~is~unknown~for~a~rule.\\
9415   \c_@@_available_keys_str
9416 }
9417 {
9418   The~available~keys~are~(in~alphabetic~order):~
9419   color,~
9420   dotted,~
9421   multiplicity,~
9422   sep-color,~
9423   tikz,~and~total-width.
9424 }
9425
9426 \@@_msg_new:nnn { Unknown~key~for~Block }
9427 {
9428   Unknown~key.\\
9429   The~key~'\l_keys_key_str'~is~unknown~for~the~command~\token_to_str:N
9430   \Block.\\ It~will~be~ignored. \\
9431   \c_@@_available_keys_str
9432 }
9433 {
9434   The~available~keys~are~(in~alphabetic~order):~b,~B,~borders,~c,~draw,~fill,~
9435   hlines,~hvlines,~l,~line-width,~name,~opacity,~rounded-corners,~r,~
9436   respect-arraystretch,~t,~T,~tikz,~transparent~and~vlines.
9437 }
9438 \@@_msg_new:nn { Version~of~siunitx~too~old }
9439 {
9440   siunitx~too~old.\\
9441   You~can't~use~'S'~columns~because~your~version~of~'siunitx'~
9442   is~too~old.~You~need~at~least~v3.0.38~and~your~log~file~says:~"siunitx,~
9443   \use:c { ver @ siunitx.sty }". \\
9444   This~error~is~fatal.
9445 }
9446 \@@_msg_new:nnn { Unknown~key~for~Brace }

```

```

9447 {
9448   Unknown~key.\\
9449   The~key~'\l_keys_key_str'~is~unknown~for~the~commands~\token_to_str:N
9450   \UnderBrace\ and~\token_to_str:N \OverBrace.\\
9451   It~will~be~ignored. \\
9452   \c_@@_available_keys_str
9453 }
9454 {
9455   The~available~keys~are~(in~alphabetic~order):~color,~left~shorten,~
9456   right~shorten,~shorten~(which~fixes~both~left~shorten~and~
9457   right~shorten)~and~yshift.
9458 }
9459 \@@_msg_new:nnn { Unknown~key~for~CodeAfter }
9460 {
9461   Unknown~key.\\
9462   The~key~'\l_keys_key_str'~is~unknown.\\
9463   It~will~be~ignored. \\
9464   \c_@@_available_keys_str
9465 }
9466 {
9467   The~available~keys~are~(in~alphabetic~order):~
9468   delimiters/color,~
9469   rules~(with~the~subkeys~'color'~and~'width'),~
9470   sub-matrix~(several~subkeys)~
9471   and~\dots~(several~subkeys).~
9472   The~latter~is~for~the~command~\token_to_str:N \line.
9473 }
9474 \@@_msg_new:nnn { Unknown~key~for~CodeBefore }
9475 {
9476   Unknown~key.\\
9477   The~key~'\l_keys_key_str'~is~unknown.\\
9478   It~will~be~ignored. \\
9479   \c_@@_available_keys_str
9480 }
9481 {
9482   The~available~keys~are~(in~alphabetic~order):~
9483   create-cell-nodes,~
9484   delimiters/color~and~
9485   sub-matrix~(several~subkeys).
9486 }
9487 \@@_msg_new:nnn { Unknown~key~for~SubMatrix }
9488 {
9489   Unknown~key.\\
9490   The~key~'\l_keys_key_str'~is~unknown.\\
9491   That~key~will~be~ignored. \\
9492   \c_@@_available_keys_str
9493 }
9494 {
9495   The~available~keys~are~(in~alphabetic~order):~
9496   'delimiters/color',~
9497   'extra-height',~
9498   'hlines',~
9499   'hvlines',~
9500   'left-xshift',~
9501   'name',~
9502   'right-xshift',~
9503   'rules'~(with~the~subkeys~'color'~and~'width'),~
9504   'slim',~
9505   'vlines'~and~'xshift'~(which~sets~both~'left-xshift'~
9506   and~'right-xshift').\\
9507 }
9508 \@@_msg_new:nnn { Unknown~key~for~notes }

```

```

9509 {
9510     Unknown~key.\\
9511     The~key~'\l_keys_key_str'~is~unknown.\\
9512     That~key~will~be~ignored. \\
9513     \c_@@_available_keys_str
9514 }
9515 {
9516     The~available~keys~are~(in~alphabetic~order):~
9517     bottomrule,~
9518     code~after,~
9519     code~before,~
9520     detect~duplicates,~
9521     enumitem~keys,~
9522     enumitem~keys~para,~
9523     para,~
9524     label~in~list,~
9525     label~in~tabular~and~
9526     style.
9527 }
9528 \@@_msg_new:nnn { Unknown~key~for~RowStyle }
9529 {
9530     Unknown~key.\\
9531     The~key~'\l_keys_key_str'~is~unknown~for~the~command~
9532     \token_to_str:N \RowStyle. \\
9533     That~key~will~be~ignored. \\
9534     \c_@@_available_keys_str
9535 }
9536 {
9537     The~available~keys~are~(in~alphabetic~order):~
9538     'bold',~
9539     'cell-space-top-limit',~
9540     'cell-space-bottom-limit',~
9541     'cell-space-limits',~
9542     'color',~
9543     'nb-rows'~and~
9544     'rowcolor'.
9545 }
9546 \@@_msg_new:nnn { Unknown~key~for~NiceMatrixOptions }
9547 {
9548     Unknown~key.\\
9549     The~key~'\l_keys_key_str'~is~unknown~for~the~command~
9550     \token_to_str:N \NiceMatrixOptions. \\
9551     That~key~will~be~ignored. \\
9552     \c_@@_available_keys_str
9553 }
9554 {
9555     The~available~keys~are~(in~alphabetic~order):~
9556     allow~duplicate~names,~
9557     caption~above,~
9558     cell~space~bottom~limit,~
9559     cell~space~limits,~
9560     cell~space~top~limit,~
9561     code~for~first~col,~
9562     code~for~first~row,~
9563     code~for~last~col,~
9564     code~for~last~row,~
9565     corners,~
9566     custom~key,~
9567     create~extra~nodes,~
9568     create~medium~nodes,~
9569     create~large~nodes,~
9570     delimiters~(several~subkeys),~
9571     end~of~row,~

```

```

9572 first-col,~
9573 first-row,~
9574 hlines,~
9575 hvlines,~
9576 hvlines-except-borders,~
9577 last-col,~
9578 last-row,~
9579 left-margin,~
9580 light-syntax,~
9581 matrix/columns-type,~
9582 notes~(several-subkeys),~
9583 nullify-dots,~
9584 pgf-node-code,~
9585 renew-dots,~
9586 renew-matrix,~
9587 respect-arraystretch,~
9588 rounded-corners,~
9589 right-margin,~
9590 rules~(with-the-subkeys~'color'~and~'width'),~
9591 small,~
9592 sub-matrix~(several-subkeys),~
9593 vlines,~
9594 xdots~(several-subkeys).
9595 }

```

For '{NiceArray}', the set of keys is the same as for {NiceMatrix} excepted that there is no l and r.

```

9596 \@@_msg_new:nnn { Unknown~key~for~NiceArray }
9597 {
9598   Unknown~key.\\
9599   The~key~'\l_keys_key_str'~is~unknown~for~the~environment~
9600   \{NiceArray\}. \\
9601   That~key~will~be~ignored. \\
9602   \c_@@_available_keys_str
9603 }
9604 {
9605   The~available~keys~are~(in~alphabetic~order):~
9606   b,~
9607   baseline,~
9608   c,~
9609   cell-space-bottom-limit,~
9610   cell-space-limits,~
9611   cell-space-top-limit,~
9612   code-after,~
9613   code-for-first-col,~
9614   code-for-first-row,~
9615   code-for-last-col,~
9616   code-for-last-row,~
9617   color-inside,~
9618   columns-width,~
9619   corners,~
9620   create-extra-nodes,~
9621   create-medium-nodes,~
9622   create-large-nodes,~
9623   extra-left-margin,~
9624   extra-right-margin,~
9625   first-col,~
9626   first-row,~
9627   hlines,~
9628   hvlines,~
9629   hvlines-except-borders,~
9630   last-col,~
9631   last-row,~
9632   left-margin,~

```

```

9633 light-syntax,~
9634 name,~
9635 nullify-dots,~
9636 pgf-node-code,~
9637 renew-dots,~
9638 respect-arraystretch,~
9639 right-margin,~
9640 rounded-corners,~
9641 rules~(with~the~subkeys~'color'~and~'width'),~
9642 small,~
9643 t,~
9644 vl原因,~
9645 xdots/color,~
9646 xdots/shorten-start,~
9647 xdots/shorten-end,~
9648 xdots/shorten-and~
9649 xdots/line-style.
9650 }

```

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).

```

9651 \@@_msg_new:nnn { Unknown~key~for~NiceMatrix }
9652 {
9653   Unknown~key.\\
9654   The~key~'\l_keys_key_str'~is~unknown~for~the~
9655   \@@_full_name_env:. \\
9656   That~key~will~be~ignored. \\
9657   \c_@@_available_keys_str
9658 }
9659 {
9660   The~available~keys~are~(in~alphabetic~order):~
9661   b,~
9662   baseline,~
9663   c,~
9664   cell-space-bottom-limit,~
9665   cell-space-limits,~
9666   cell-space-top-limit,~
9667   code-after,~
9668   code-for-first-col,~
9669   code-for-first-row,~
9670   code-for-last-col,~
9671   code-for-last-row,~
9672   color-inside,~
9673   columns-type,~
9674   columns-width,~
9675   corners,~
9676   create-extra-nodes,~
9677   create-medium-nodes,~
9678   create-large-nodes,~
9679   extra-left-margin,~
9680   extra-right-margin,~
9681   first-col,~
9682   first-row,~
9683   hlines,~
9684   hvlines,~
9685   hvlines-except-borders,~
9686   l,~
9687   last-col,~
9688   last-row,~
9689   left-margin,~
9690   light-syntax,~
9691   name,~
9692   nullify-dots,~
9693   pgf-node-code,~

```

```

9694     r,~
9695     renew-dots,~
9696     respect-arraystretch,~
9697     right-margin,~
9698     rounded-corners,~
9699     rules~(with~the~subkeys~'color'~and~'width'),~
9700     small,~
9701     t,~
9702     vlines,~
9703     xdots/color,~
9704     xdots/shorten-start,~
9705     xdots/shorten-end,~
9706     xdots/shorten~and~
9707     xdots/line-style.
9708 }
9709 \@@_msg_new:nnn { Unknown~key~for~NiceTabular }
9710 {
9711     Unknown~key.\\
9712     The~key~'\l_keys_key_str'~is~unknown~for~the~environment~
9713     \{NiceTabular\}. \\
9714     That~key~will~be~ignored. \\
9715     \c_@@_available_keys_str
9716 }
9717 {
9718     The~available~keys~are~(in~alphabetic~order):~
9719     b,~
9720     baseline,~
9721     c,~
9722     caption,~
9723     cell-space-bottom-limit,~
9724     cell-space-limits,~
9725     cell-space-top-limit,~
9726     code-after,~
9727     code-for-first-col,~
9728     code-for-first-row,~
9729     code-for-last-col,~
9730     code-for-last-row,~
9731     color-inside,~
9732     columns-width,~
9733     corners,~
9734     custom-line,~
9735     create-extra-nodes,~
9736     create-medium-nodes,~
9737     create-large-nodes,~
9738     extra-left-margin,~
9739     extra-right-margin,~
9740     first-col,~
9741     first-row,~
9742     hlines,~
9743     hvlines,~
9744     hvlines-except-borders,~
9745     label,~
9746     last-col,~
9747     last-row,~
9748     left-margin,~
9749     light-syntax,~
9750     name,~
9751     notes~(several~subkeys),~
9752     nullify-dots,~
9753     pgf-node-code,~
9754     renew-dots,~
9755     respect-arraystretch,~
9756     right-margin,~

```

```

9757 rounded-corners,~
9758 rules~(with~the~subkeys~'color'~and~'width'),~
9759 short-caption,~
9760 t,~
9761 tabularnote,~
9762 vlines,~
9763 xdots/color,~
9764 xdots/shorten-start,~
9765 xdots/shorten-end,~
9766 xdots/shorten~and~
9767 xdots/line-style.
9768 }

9769 \@@_msg_new:nnn { Duplicate-name }
9770 {
9771   Duplicate-name.\\
9772   The~name~'\l_keys_value_tl'~is~already~used~and~you~shouldn't~use~
9773   the~same~environment~name~twice.~You~can~go~on,~but,~
9774   maybe,~you~will~have~incorrect~results~especially~
9775   if~you~use~'columns-width=auto'.~If~you~don't~want~to~see~this~
9776   message~again,~use~the~key~'allow-duplicate-names'~in~
9777   '\token_to_str:N \NiceMatrixOptions'.\\
9778   \bool_if:NF \g_@@_messages_for_Overleaf_bool
9779     { For~a~list~of~the~names~already~used,~type~H~<return>. }
9780 }
9781 {
9782   The~names~already~defined~in~this~document~are:~
9783   \seq_use:Nnnn \g_@@_names_seq { ~and~ } { ,~ } { ~and~ }.
9784 }

9785 \@@_msg_new:nn { Option~auto~for~columns~width }
9786 {
9787   Erroneous-use.\\
9788   You~can't~give~the~value~'auto'~to~the~key~'columns-width'~here.~
9789   That~key~will~be~ignored.
9790 }

9791 \@@_msg_new:nn { NiceTabularX~without~X }
9792 {
9793   NiceTabularX~without~X.\\
9794   You~should~not~use~{NiceTabularX}~without~X~columns.\\
9795   However,~you~can~go~on.
9796 }

```

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>	70
14	The environment <code>{NiceMatrix}</code> and its variants	88
15	<code>{NiceTabular}</code> , <code>{NiceTabularX}</code> and <code>{NiceTabular*}</code>	88
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	109
19	User commands available in the new environments	115
20	The command <code>\line</code> accessible in code-after	121
21	The command <code>\RowStyle</code>	123
22	Colors of cells, rows and columns	125
23	The vertical and horizontal rules	134
24	The key corners	149
25	The environment <code>{NiceMatrixBlock}</code>	152
26	The extra nodes	153
27	The blocks	157
28	How to draw the dotted lines transparently	176
29	Automatic arrays	177
30	The redefinition of the command <code>\dotfill</code>	178

31	The command <code>\diagbox</code>	179
32	The keyword <code>\CodeAfter</code>	180
33	The delimiters in the preamble	181
34	The command <code>\SubMatrix</code>	182
35	Les commandes <code>\UnderBrace</code> et <code>\OverBrace</code>	190
36	The command <code>TikzEveryCell</code>	193
37	The command <code>\ShowCellNames</code>	195
38	We process the options at package loading	197
39	About the package underscore	199
40	Error messages of the package	199