The code of the package **nicematrix**[*]

F. Pantigny
`fpantigny@wanadoo.fr`

October 7, 2024

Abstract

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

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

1 Declaration of the package and packages loaded

The prefix `nicematrix` has been 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.

```
9 \RequirePackage { amsmath }
```

```
10 \RequirePackage { array }
```

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

```
11 \bool_const:Nn \c_@@_tagging_array_bool
12   { \IfPackageAtLeastTF { array } { 2024/05/01 } \c_true_bool \c_false_bool }
13 \bool_const:Nn \c_@@_testphase_table_bool
14   { \IfPackageLoadedTF { latex-lab-testphase-table } \c_true_bool \c_false_bool }
15   }
```

---

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

```
16  \cs_new_protected:Npn \@@_error:n { \msg_error:nn { nicematrix } }
17  \cs_new_protected:Npn \@@_warning:n { \msg_warning:nn { nicematrix } }
18  \cs_new_protected:Npn \@@_error:nn { \msg_error:nnn { nicematrix } }
19  \cs_generate_variant:Nn \@@_error:nn { n e }
20  \cs_new_protected:Npn \@@_error:nnn { \msg_error:nnnn { nicematrix } }
21  \cs_new_protected:Npn \@@_fatal:n { \msg_fatal:nn { nicematrix } }
22  \cs_new_protected:Npn \@@_fatal:nn { \msg_fatal:nnn { nicematrix } }
23  \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).

```
24  \cs_new_protected:Npn \@@_msg_new:nnn #1 #2 #3
25    {
26      \bool_if:NTF \g_@@_messages_for_Overleaf_bool
27        { \msg_new:nnn { nicematrix } { #1 } { #2 \\ #3 } }
28        { \msg_new:nnnn { nicematrix } { #1 } { #2 } { #3 } }
29    }
```

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

```
30  \cs_new_protected:Npn \@@_error_or_warning:n
31    { \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".

```
32  \bool_new:N \g_@@_messages_for_Overleaf_bool
33  \bool_gset:Nn \g_@@_messages_for_Overleaf_bool
34    {
35        \str_if_eq_p:on \c_sys_jobname_str { _region_ }  % for Emacs
36      || \str_if_eq_p:on \c_sys_jobname_str { output }   % for Overleaf
37    }
```

```
38  \cs_new_protected:Npn \@@_msg_redirect_name:nn
39    { \msg_redirect_name:nnn { nicematrix } }
40  \cs_new_protected:Npn \@@_gredirect_none:n #1
41    {
42      \group_begin:
43      \globaldefs = 1
44      \@@_msg_redirect_name:nn { #1 } { none }
45      \group_end:
46    }
47  \cs_new_protected:Npn \@@_err_gredirect_none:n #1
48    {
49      \@@_error:n { #1 }
50      \@@_gredirect_none:n { #1 }
51    }
52  \cs_new_protected:Npn \@@_warning_gredirect_none:n #1
53    {
54      \@@_warning:n { #1 }
55      \@@_gredirect_none:n { #1 }
56    }
```

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

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

2 Security test

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

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

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

```
59  \@@_msg_new:nn { Internal~error }
60    {
61      Potential~problem~when~using~nicematrix.\\
62      The~package~nicematrix~have~detected~a~modification~of~the~
63      standard~environment~{array}~(of~the~package~array).~Maybe~you~will~encounter~
64      some~slight~problems~when~using~nicematrix.~If~you~don't~want~to~see~
65      this~message~again,~load~nicematrix~with:~\token_to_str:N
66      \usepackage[no-test-for-array]{nicematrix}.
67    }
```

```
68  \@@_msg_new:nn { mdwtab~loaded }
69    {
70      The~packages~'mdwtab'~and~'nicematrix'~are~incompatible.~
71      This~error~is~fatal.
72    }
```

```
73  \cs_new_protected:Npn \@@_security_test:n #1
74    {
75      \peek_meaning:NTF \ignorespaces
76        { \@@_security_test_i:w }
77        { \@@_error:n { Internal~error } }
78      #1
79    }
```

```
80  \bool_if:NTF \c_@@_tagging_array_bool
81    {
82      \cs_new_protected:Npn \@@_security_test_i:w \ignorespaces #1
83        {
84          \peek_meaning:NF \textonly@unskip { \@@_error:n { Internal~error } }
85          #1
86        }
87    }
88    {
89      \cs_new_protected:Npn \@@_security_test_i:w \ignorespaces #1
90        {
91          \peek_meaning:NF \unskip { \@@_error:n { Internal~error } }
92          #1
93        }
94    }
```

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

```
95  \hook_gput_code:nnn { begindocument / end } { . }
96    {
97      \IfPackageLoadedTF { mdwtab }
98        { \@@_fatal:n { mdwtab~loaded } }
99        {
100         \bool_if:NF \g_@@_no_test_for_array_bool
```

```
101           {
102             \group_begin:
103               \hbox_set:Nn \l_tmpa_box
104                 {
105                   \begin { tabular } { c > { \@@_security_test:n } c c }
106                   text & & text
107                   \end { tabular }
108                 }
109             \group_end:
110           }
111       }
112   }
```

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`).

```
113 \cs_new_protected:Npn \@@_collect_options:n #1
114   {
115     \peek_meaning:NTF [
116       { \@@_collect_options:nw { #1 } }
117       { #1 { } }
118   }
```

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

```
119 \NewDocumentCommand \@@_collect_options:nw { m r[] }
120   { \@@_collect_options:nn { #1 } { #2 } }
121
122 \cs_new_protected:Npn \@@_collect_options:nn #1 #2
123   {
124     \peek_meaning:NTF [
125       { \@@_collect_options:nnw { #1 } { #2 } }
126       { #1 { #2 } }
127   }
128
129 \cs_new_protected:Npn \@@_collect_options:nnw #1#2[#3]
130   { \@@_collect_options:nn { #1 } { #2 , #3 } }
```

4 Technical definitions

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

```
131 \tl_const:Nn \c_@@_b_tl { b }
132 \tl_const:Nn \c_@@_c_tl { c }
133 \tl_const:Nn \c_@@_l_tl { l }
```

```
134  \tl_const:Nn \c_@@_r_tl { r }
135  \tl_const:Nn \c_@@_all_tl { all }
136  \tl_const:Nn \c_@@_dot_tl { . }
137  \tl_const:Nn \c_@@_default_tl { default }
138  \tl_const:Nn \c_@@_star_tl { * }
139  \str_const:Nn \c_@@_star_str { * }
140  \str_const:Nn \c_@@_r_str { r }
141  \str_const:Nn \c_@@_c_str { c }
142  \str_const:Nn \c_@@_l_str { l }
143  \str_const:Nn \c_@@_R_str { R }
144  \str_const:Nn \c_@@_C_str { C }
145  \str_const:Nn \c_@@_L_str { L }
146  \str_const:Nn \c_@@_j_str { j }
147  \str_const:Nn \c_@@_si_str { si }
```

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

```
148  \tl_new:N \l_@@_argspec_tl

149  \cs_generate_variant:Nn \seq_set_split:Nnn { N o n }
150  \cs_generate_variant:Nn \str_lowercase:n { o }


151  \hook_gput_code:nnn { begindocument } { . }
152    {
153      \IfPackageLoadedTF { tikz }
154        {
```

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

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

```
155          \tl_const:Nn \c_@@_pgfortikzpicture_tl { \exp_not:N \tikzpicture }
156          \tl_const:Nn \c_@@_endpgfortikzpicture_tl { \exp_not:N \endtikzpicture }
157        }
158        {
159          \tl_const:Nn \c_@@_pgfortikzpicture_tl { \exp_not:N \pgfpicture }
160          \tl_const:Nn \c_@@_endpgfortikzpicture_tl { \exp_not:N \endpgfpicture }
161        }
162    }
```

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 programmation. At the date April 2024, the current version revtex4-2 is 4.2f (compatible with booktabs).

```
163  \IfClassLoadedTF { revtex4-1 }
164    { \bool_const:Nn \c_@@_revtex_bool \c_true_bool }
165    {
166      \IfClassLoadedTF { revtex4-2 }
167        { \bool_const:Nn \c_@@_revtex_bool \c_true_bool }
168        {
```

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

```
169          \cs_if_exist:NT \rvtx@ifformat@geq
170            { \bool_const:Nn \c_@@_revtex_bool \c_true_bool }
171            { \bool_const:Nn \c_@@_revtex_bool \c_false_bool }
172        }
173    }
```

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.

```
174 \cs_new_protected:Npn \@@_provide_pgfsyspdfmark:
175   {
176     \iow_now:Nn \@mainaux
177       {
178         \ExplSyntaxOn
179         \cs_if_free:NT \pgfsyspdfmark
180           { \cs_set_eq:NN \pgfsyspdfmark \@gobblethree }
181         \ExplSyntaxOff
182       }
183     \cs_gset_eq:NN \@@_provide_pgfsyspdfmark: \prg_do_nothing:
184   }
```

We define a command \iddots similar to \ddots (⋱) but with dots going forward (⋰). 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.

```
185 \ProvideDocumentCommand \iddots { }
186   {
187     \mathinner
188       {
189         \tex_mkern:D 1 mu
190         \box_move_up:nn { 1 pt } { \hbox { . } }
191         \tex_mkern:D 2 mu
192         \box_move_up:nn { 4 pt } { \hbox { . } }
193         \tex_mkern:D 2 mu
194         \box_move_up:nn { 7 pt }
195           { \vbox:n { \kern 7 pt \hbox { . } } }
196         \tex_mkern:D 1 mu
197       }
198   }
```

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.

```
199 \hook_gput_code:nnn { begindocument } { . }
200   {
201     \IfPackageLoadedTF { booktabs }
202       { \iow_now:Nn \@mainaux \nicematrix@redefine@check@rerun }
203       { }
204   }
205 \cs_set_protected:Npn \nicematrix@redefine@check@rerun
206   {
207     \cs_set_eq:NN \@@_old_pgfutil@check@rerun \pgfutil@check@rerun
```

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

```
208     \cs_set_protected:Npn \pgfutil@check@rerun ##1 ##2
209       {
210         \str_if_eq:eeF { nm- } { \tl_range:nnn { ##1 } 1 3 }
211           { \@@_old_pgfutil@check@rerun { ##1 } { ##2 } }
212       }
213   }
```

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

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

```
215    {
216      \IfPackageLoadedTF { colortbl }
217        { }
218        {
```

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.

```
219          \cs_set_protected:Npn \CT@arc@ { }
220          \cs_set_nopar:Npn \arrayrulecolor #1 # { \CT@arc { #1 } }
221          \cs_set_nopar:Npn \CT@arc #1 #2
222            {
223              \dim_compare:nNnT \baselineskip = \c_zero_dim \noalign
224                { \cs_gset_nopar:Npn \CT@arc@ { \color #1 { #2 } } }
225            }
```

Idem for `\CT@drs@`.

```
226          \cs_set_nopar:Npn \doublerulesepcolor #1 # { \CT@drs { #1 } }
227          \cs_set_nopar:Npn \CT@drs #1 #2
228            {
229              \dim_compare:nNnT \baselineskip = \c_zero_dim \noalign
230                { \cs_gset:Npn \CT@drsc@ { \color #1 { #2 } } }
231            }
232          \cs_set_nopar:Npn \hline
233            {
234              \noalign { \ifnum 0 = `} \fi
235              \cs_set_eq:NN \hskip \vskip
236              \cs_set_eq:NN \vrule \hrule
237              \cs_set_eq:NN \@width \@height
238              { \CT@arc@ \vline }
239              \futurelet \reserved@a
240              \@xhline
241            }
242        }
243    }
```

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.

```
244 \cs_set_nopar:Npn \@@_standard_cline #1 { \@@_standard_cline:w #1 \q_stop }
245 \cs_set_nopar:Npn \@@_standard_cline:w #1-#2 \q_stop
246   {
247     \int_if_zero:nT \l_@@_first_col_int { \omit & }
248     \int_compare:nNnT { #1 } > \c_one_int
249       { \multispan { \int_eval:n { #1 - 1 } } & }
250     \multispan { \int_eval:n { #2 - #1 + 1 } }
251     {
252       \CT@arc@
253       \leaders \hrule \@height \arrayrulewidth \hfill
```

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

```
254       \skip_horizontal:N \c_zero_dim
255     }
```

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`.

```
256     \everycr { }
257     \cr
258     \noalign { \skip_vertical:N -\arrayrulewidth }
259   }
```

---

[1] See question 99041 on TeX StackExchange.

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.

```
260 \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`.

```
261   { \@@_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*.

```
262 \cs_set:Npn \@@_cline_i:nn #1 #2 { \@@_cline_i:w #1|#2- \q_stop }
263 \cs_set:Npn \@@_cline_i:w #1|#2-#3 \q_stop
264   {
265     \tl_if_empty:nTF { #3 }
266       { \@@_cline_iii:w #1|#2-#2 \q_stop }
267       { \@@_cline_ii:w #1|#2-#3 \q_stop }
268   }
269 \cs_set:Npn \@@_cline_ii:w #1|#2-#3-\q_stop
270   { \@@_cline_iii:w #1|#2-#3 \q_stop }
271 \cs_set:Npn \@@_cline_iii:w #1|#2-#3 \q_stop
272   {
```

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).

```
273     \int_compare:nNnT { #1 } < { #2 }
274       { \multispan { \int_eval:n { #2 - #1 } } & }
275     \multispan { \int_eval:n { #3 - #2 + 1 } }
276       {
277         \CT@arc@
278         \leaders \hrule \@height \arrayrulewidth \hfill
279         \skip_horizontal:N \c_zero_dim
280       }
```

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

```
281     \peek_meaning_remove_ignore_spaces:NTF \cline
282       { & \@@_cline_i:en { \int_eval:n { #3 + 1 } } }
283       { \everycr { } \cr }
284   }
285 \cs_generate_variant:Nn \@@_cline_i:nn { e n }
```

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

```
286 \cs_set_eq:NN \@@_math_toggle: \c_math_toggle_token
```

```
287 \cs_new_protected:Npn \@@_set_CT@arc@:n #1
288   {
289     \tl_if_blank:nF { #1 }
290       {
291         \tl_if_head_eq_meaning:nNTF { #1 } [
292           { \cs_set_nopar:Npn \CT@arc@ { \color #1 } }
293           { \cs_set_nopar:Npn \CT@arc@ { \color { #1 } } }
294       }
295   }
296 \cs_generate_variant:Nn \@@_set_CT@arc@:n { o }
```

```
297 \cs_new_protected:Npn \@@_set_CT@drsc@:n #1
298   {
299     \tl_if_head_eq_meaning:nNTF { #1 } [
300       { \cs_set_nopar:Npn \CT@drsc@ { \color #1 } }
301       { \cs_set_nopar:Npn \CT@drsc@ { \color { #1 } } }
302   }
303 \cs_generate_variant:Nn \@@_set_CT@drsc@:n { o }
```

8

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

```
304 \cs_new:Npn \@@_exp_color_arg:Nn #1 #2
305   {
306     \tl_if_head_eq_meaning:nNTF { #2 } [
307        { #1 #2 }
308        { #1 { #2 } }
309   }
310 \cs_generate_variant:Nn \@@_exp_color_arg:Nn { N o }
```

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

```
311 \cs_new_protected:Npn \@@_color:n #1
312   { \tl_if_blank:nF { #1 } { \@@_exp_color_arg:Nn \color { #1 } } }
313 \cs_generate_variant:Nn \@@_color:n { o }
```

```
314 \cs_new_protected:Npn \@@_rescan_for_spanish:N #1
315   {
316     \tl_set_rescan:Nno
317       #1
318       {
319         \char_set_catcode_other:N >
320         \char_set_catcode_other:N <
321       }
322       #1
323   }
```

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.

```
324 \int_new:N \g_@@_env_int
```

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

```
325 \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.

```
326 \NewExpandableDocumentCommand \NiceMatrixLastEnv { }
327   { \int_use:N \g_@@_env_int }
```

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

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

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

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

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

```
331 \bool_new:N \g_@@_delims_bool
332 \bool_gset_true:N \g_@@_delims_bool
```

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

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

```
333 \bool_new:N \l_@@_preamble_bool
334 \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.

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

The following counter will count the environments {NiceMatrixBlock}.

```
336 \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.

```
337 \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).

```
338 \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.).

```
339 \dim_new:N \l_@@_col_width_dim
340 \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.

```
341 \int_new:N \g_@@_row_total_int
342 \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).

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

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

```
344 \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[l]{3cm}` will provide the value `l` for all the cells of the column.

```
345 \tl_new:N \l_@@_hpos_cell_tl
346 \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_c_tl
```

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

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

Idem for the mono-row blocks.

348 `\dim_new:N \g_@@_blocks_ht_dim`
349 `\dim_new:N \g_@@_blocks_dp_dim`

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

350 `\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`.

351 `\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.

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

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

353 `\bool_new:N \l_@@_notes_detect_duplicates_bool`
354 `\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.

355 `\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).

356 `\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).

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

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

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

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

359 `\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.

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

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

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

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

11

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

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

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

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

```
365  \tl_new:N \g_@@_left_delim_tl
366  \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}`).

```
367  \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`).

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

For `\multicolumn`.

```
369  \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`.

```
370  \tl_new:N \l_@@_columns_type_tl
371  \str_set:Nn \l_@@_columns_type_tl { c }
```

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

```
372  \tl_new:N \l_@@_xdots_down_tl
373  \tl_new:N \l_@@_xdots_up_tl
374  \tl_new:N \l_@@_xdots_middle_tl
```

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

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

```
376  \cs_new_protected:Npn \@@_test_if_math_mode:
377    {
378      \if_mode_math: \else:
379        \@@_fatal:n { Outside~math~mode }
380      \fi:
381    }
```

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.

```
382  \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".

```
383  \colorlet { nicematrix-last-col } { . }
384  \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*).

```
385  \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*.

```
386 \tl_new:N \g_@@_com_or_env_str
387 \tl_gset:Nn \g_@@_com_or_env_str { environment }
```

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

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

```
389 \cs_new:Npn \@@_full_name_env:
390   {
391     \str_if_eq:onTF \g_@@_com_or_env_str { command }
392       { command \space \c_backslash_str \g_@@_name_env_str }
393       { environment \space \{ \g_@@_name_env_str \} }
394   }
```

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

```
395 \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.

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

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

```
397 \tl_new:N \g_@@_pre_code_before_tl
398 \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.

```
399 \tl_new:N \g_@@_pre_code_after_tl
400 \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.

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

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

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

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

```
403 \int_new:N \l_@@_old_iRow_int
404 \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).

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

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

```
406 \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.

```
407 \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 weigth 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.

```
408 \bool_new:N \l_@@_X_columns_aux_bool
409 \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.

```
410 \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).

```
411 \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).

```
412 \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`).

```
413 \tl_new:N \l_@@_code_before_tl
414 \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).

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

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

```
416 \dim_new:N \l_@@_x_initial_dim
417 \dim_new:N \l_@@_y_initial_dim
418 \dim_new:N \l_@@_x_final_dim
419 \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.

```
420 \dim_new:N \l_@@_tmpc_dim
421 \dim_new:N \l_@@_tmpd_dim
```

```
422  \dim_new:N \g_@@_dp_row_zero_dim
423  \dim_new:N \g_@@_ht_row_zero_dim
424  \dim_new:N \g_@@_ht_row_one_dim
425  \dim_new:N \g_@@_dp_ante_last_row_dim
426  \dim_new:N \g_@@_ht_last_row_dim
427  \dim_new:N \g_@@_dp_last_row_dim
```

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

```
428  \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".

```
429  \dim_new:N \g_@@_width_last_col_dim
430  \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.

```
431  \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!).

```
432  \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*}.

```
433  \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.

```
434  \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.

```
435  \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`).

```
436  \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.

```
437  \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).

```
438 \seq_new:N \g_@@_multicolumn_cells_seq
439 \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`).

```
440 \int_new:N \l_@@_row_min_int
441 \int_new:N \l_@@_row_max_int
442 \int_new:N \l_@@_col_min_int
443 \int_new:N \l_@@_col_max_int
```

The following counters will be used when drawing the rules.

```
444 \int_new:N \l_@@_start_int
445 \int_set_eq:NN \l_@@_start_int \c_one_int
446 \int_new:N \l_@@_end_int
447 \int_new:N \l_@@_local_start_int
448 \int_new:N \l_@@_local_end_int
```

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

```
449 \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).

```
450 \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`.

```
451 \tl_new:N \l_@@_fill_tl
452 \tl_new:N \l_@@_opacity_tl
453 \tl_new:N \l_@@_draw_tl
454 \seq_new:N \l_@@_tikz_seq
455 \clist_new:N \l_@@_borders_clist
456 \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.

```
457 \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`.

```
458 \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.

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

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

```
460 \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`).

```
461 \str_new:N \l_@@_hpos_block_str
462 \str_set:Nn \l_@@_hpos_block_str { c }
463 \bool_new:N \l_@@_hpos_of_block_cap_bool
464 \bool_new:N \l_@@_p_block_bool
```

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

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

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

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

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

```
467 \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).

```
468 \bool_new:N \l_@@_vlines_block_bool
469 \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.

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

```
471 \dim_new:N \l_@@_submatrix_extra_height_dim
472 \dim_new:N \l_@@_submatrix_left_xshift_dim
473 \dim_new:N \l_@@_submatrix_right_xshift_dim
474 \clist_new:N \l_@@_hlines_clist
475 \clist_new:N \l_@@_vlines_clist
476 \clist_new:N \l_@@_submatrix_hlines_clist
477 \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}`).

```
478 \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.

```
479 \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`).

```
480 \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.

```
481    \int_new:N \l_@@_first_row_int
482    \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.

  ```
  483    \int_new:N \l_@@_first_col_int
  484    \int_set_eq:NN \l_@@_first_col_int \c_one_int
  ```

- **Last row**

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

  ```
  485    \int_new:N \l_@@_last_row_int
  486    \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]

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

  Idem for `\l_@@_last_col_without_value_bool`

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

- **Last column**

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

  ```
  489    \int_new:N \l_@@_last_col_int
  490    \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.

  ```
  491    \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`).

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

---

[2]We 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.

Some utilities

```
493  \cs_set_protected:Npn \@@_cut_on_hyphen:w #1-#2\q_stop
494    {
495      \cs_set_nopar:Npn \l_tmpa_tl { #1 }
496      \cs_set_nopar:Npn \l_tmpb_tl { #2 }
497    }
```

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.

```
498  \cs_new_protected:Npn \@@_expand_clist:N #1
499    {
500      \clist_if_in:NVF #1 \c_@@_all_tl
501        {
502          \clist_clear:N \l_tmpa_clist
503          \clist_map_inline:Nn #1
504            {
505              \tl_if_in:nnTF { ##1 } { - }
506                { \@@_cut_on_hyphen:w ##1 \q_stop }
507                {
508                  \cs_set_nopar:Npn \l_tmpa_tl { ##1 }
509                  \cs_set_nopar:Npn \l_tmpb_tl { ##1 }
510                }
511              \int_step_inline:nnn \l_tmpa_tl \l_tmpb_tl
512                { \clist_put_right:Nn \l_tmpa_clist { ####1 } }
513            }
514          \tl_set_eq:NN #1 \l_tmpa_clist
515        }
516    }
```

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.

```
517  \hook_gput_code:nnn { begindocument } { . }
518    {
519      \dim_const:Nn \c_@@_shift_Ldots_last_row_dim { 0.5 em }
520      \dim_const:Nn \c_@@_shift_exterior_Vdots_dim { 0.6 em }
521      \dim_const:Nn \c_@@_innersep_middle_dim { 0.17 em }
522    }
```

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 ot the tabular). However, we want the labels of the successive tabular notes in the logical order. That's why:
    - The number of tabular notes present in the caption will be written on the `aux` file and available in `\g_@@_notes_caption_int`.[3]
    - During the composition of the main tabular, the tabular notes will be numbered from `\g_@@_notes_caption_int`+1 and the notes will be stored in `\g_@@_notes_seq`. Each component of `\g_@@_notes_seq` will be a kind of couple of the form : {*label*}{*text of the tabularnote*}. The first component is the optional argument (between square brackets) of the command `\tabularnote` (if the optional argument is not used, the value will be the special marker expressed by `\c_novalue_tl`).
    - During the composition of the caption (value of `\l_@@_caption_tl`), the tabular notes will be numbered from 1 to `\g_@@_notes_caption_int` and the notes themselves will be stored in `\g_@@_notes_in_caption_seq`. The structure of the components of that sequence will be the same as for `\g_@@_notes_seq`.
    - After the composition of the main tabular and after the composition of the caption, the sequences `\g_@@_notes_in_caption_seq` and `\g_@@_notes_seq` will be merged (in that order) and the notes will be composed.

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

```
523 \newcounter { tabularnote }
```

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

```
524 \int_new:N \g_@@_tabularnote_int
525 \cs_set:Npn \theHtabularnote { \int_use:N \g_@@_tabularnote_int }

526 \seq_new:N \g_@@_notes_seq
527 \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.

```
528 \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.

```
529 \seq_new:N \l_@@_notes_labels_seq
530 \newcounter{nicematrix_draft}
531 \cs_new_protected:Npn \@@_notes_format:n #1
532   {
533     \setcounter { nicematrix_draft } { #1 }
534     \@@_notes_style:n { nicematrix_draft }
535   }
```

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

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

---

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

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

```
537 \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`.

```
538 \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`.

```
539 \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`).

```
540 \hook_gput_code:nnn { begindocument } { . }
541   {
542     \IfPackageLoadedTF { enumitem }
543       {
```

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.

```
544         \newlist { tabularnotes } { enumerate } { 1 }
545         \setlist [ tabularnotes ]
546           {
547             topsep = 0pt ,
548             noitemsep ,
549             leftmargin = * ,
550             align = left ,
551             labelsep = 0pt ,
552             label =
553               \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotesi } } ,
554           }
555         \newlist { tabularnotes* } { enumerate* } { 1 }
556         \setlist [ tabularnotes* ]
557           {
558             afterlabel = \nobreak ,
559             itemjoin = \quad ,
560             label =
561               \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotes*i } }
562           }
```

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`.

```
563         \NewDocumentCommand \tabularnote { o m }
564           {
565             \bool_lazy_or:nnT { \cs_if_exist_p:N \@captype } \l_@@_in_env_bool
566               {
567                 \bool_lazy_and:nnTF { ! \l_@@_tabular_bool } \l_@@_in_env_bool
568                   { \@@_error:n { tabularnote~forbidden } }
569                   {
570                     \bool_if:NTF \l_@@_in_caption_bool
571                       \@@_tabularnote_caption:nn
572                       \@@_tabularnote:nn
573                       { #1 } { #2 }
574                   }
575               }
```

```
576            }
577          }
578          {
579            \NewDocumentCommand \tabularnote { o m }
580              {
581                \@@_error_or_warning:n { enumitem~not~loaded }
582                \@@_gredirect_none:n { enumitem~not~loaded }
583              }
584          }
585      }
586  \cs_new_protected:Npn \@@_test_first_novalue:nnn #1 #2 #3
587    { \tl_if_novalue:nT { #1 } { #3 } }
```

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

```
588  \cs_new_protected:Npn \@@_tabularnote:nn #1 #2
589    {
```

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).

```
590        \int_zero:N \l_tmpa_int
591        \bool_if:NT \l_@@_notes_detect_duplicates_bool
592          {
```

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

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

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

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

```
593            \int_zero:N \l_tmpb_int
594            \seq_map_indexed_inline:Nn \g_@@_notes_seq
595              {
596                \@@_test_first_novalue:nnn ##2 { \int_incr:N \l_tmpb_int }
597                \tl_if_eq:nnT { { #1 } { #2 } } { ##2 }
598                  {
599                    \tl_if_novalue:nTF { #1 }
600                      { \int_set_eq:NN \l_tmpa_int \l_tmpb_int }
601                      { \int_set:Nn \l_tmpa_int { ##1 }  }
602                    \seq_map_break:
603                  }
604              }
605            \int_if_zero:nF \l_tmpa_int
606              { \int_add:Nn \l_tmpa_int \g_@@_notes_caption_int }
607          }
608        \int_if_zero:nT \l_tmpa_int
609          {
610            \seq_gput_right:Nn \g_@@_notes_seq { { #1 } { #2 } }
611            \tl_if_novalue:nT { #1 } { \int_gincr:N \c@tabularnote }
612          }
613        \seq_put_right:Ne \l_@@_notes_labels_seq
614          {
615            \tl_if_novalue:nTF { #1 }
616              {
617                \@@_notes_format:n
618                  {
619                    \int_eval:n
```

```
620                { 
621                  \int_if_zero:nTF \l_tmpa_int
622                    \c@tabularnote
623                    \l_tmpa_int
624                }
625              }
626            }
627            { #1 }
628        }
629      \peek_meaning:NF \tabularnote
630          {
```

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

```
631            \hbox_set:Nn \l_tmpa_box
632                {
```

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

```
633              \@@_notes_label_in_tabular:n
634                {
635                  \seq_use:Nnnn
636                    \l_@@_notes_labels_seq { , } { , } { , }
637                }
638            }
```

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

```
639            \int_gdecr:N \c@tabularnote
640            \int_set_eq:NN \l_tmpa_int \c@tabularnote
```

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

```
641            \int_gincr:N \g_@@_tabularnote_int
642            \refstepcounter { tabularnote }
643            \int_compare:nNnT \l_tmpa_int = \c@tabularnote
644              { \int_gincr:N \c@tabularnote }
645            \seq_clear:N \l_@@_notes_labels_seq
646            \bool_lazy_or:nnTF
647              { \tl_if_eq_p:NN \l_@@_hpos_cell_tl \c_@@_c_tl }
648              { \tl_if_eq_p:NN \l_@@_hpos_cell_tl \c_@@_r_tl }
649              {
650                \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).

```
651                \skip_horizontal:n { \box_wd:N \l_tmpa_box }
652              }
653              { \box_use:N \l_tmpa_box }
654        }
655    }
```

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`.

```
656  \cs_new_protected:Npn \@@_tabularnote_caption:nn #1 #2
657    {
658      \bool_if:NTF \g_@@_caption_finished_bool
659        {
```

```
660        \int_compare:nNnT \c@tabularnote = \g_@@_notes_caption_int
661            { \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`!

```
662            \seq_if_in:NnF \g_@@_notes_in_caption_seq { { #1 } { #2 } }
663                { \@@_error:n { Identical~notes~in~caption } }
664        }
665        {
```

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

```
666            \seq_if_in:NnTF \g_@@_notes_in_caption_seq { { #1 } { #2 } }
667                {
```

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.

```
668                \bool_gset_true:N \g_@@_caption_finished_bool
669                \int_gset_eq:NN \g_@@_notes_caption_int \c@tabularnote
670                \int_gzero:N \c@tabularnote
671                }
672            { \seq_gput_right:Nn \g_@@_notes_in_caption_seq { { #1 } { #2 } } }
673        }
```

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!

```
674        \tl_if_novalue:nT { #1 } { \int_gincr:N \c@tabularnote }
675        \seq_put_right:Ne \l_@@_notes_labels_seq
676            {
677            \tl_if_novalue:nTF { #1 }
678                { \@@_notes_format:n { \int_use:N \c@tabularnote } }
679                { #1 }
680            }
681        \peek_meaning:NF \tabularnote
682            {
683            \@@_notes_label_in_tabular:n
684                { \seq_use:Nnnn \l_@@_notes_labels_seq { , } { , } { , } }
685            \seq_clear:N \l_@@_notes_labels_seq
686            }
687    }
688 \cs_new_protected:Npn \@@_count_novalue_first:nn #1 #2
689    { \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.

```
690 \cs_new_protected:Npn \@@_pgf_rect_node:nnnnn #1 #2 #3 #4 #5
691    {
692        \begin { pgfscope }
693        \pgfset
694            {
695            inner~sep = \c_zero_dim ,
696            minimum~size = \c_zero_dim
697            }
698        \pgftransformshift { \pgfpoint { 0.5 * ( #2 + #4 ) } { 0.5 * ( #3 + #5 ) } }
699        \pgfnode
700            { rectangle }
```

```
701        { center }
702        {
703          \vbox_to_ht:nn
704            { \dim_abs:n { #5 - #3 } }
705            {
706              \vfill
707              \hbox_to_wd:nn { \dim_abs:n { #4 - #2 } } { }
708            }
709        }
710        { #1 }
711        { }
712      \end { pgfscope }
713    }
```

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.

```
714  \cs_new_protected:Npn \@@_pgf_rect_node:nnn #1 #2 #3
715    {
716      \begin { pgfscope }
717      \pgfset
718        {
719          inner~sep = \c_zero_dim ,
720          minimum~size = \c_zero_dim
721        }
722      \pgftransformshift { \pgfpointscale { 0.5 } { \pgfpointadd { #2 } { #3 } } }
723      \pgfpointdiff { #3 } { #2 }
724      \pgfgetlastxy \l_tmpa_dim \l_tmpb_dim
725      \pgfnode
726        { rectangle }
727        { center }
728        {
729          \vbox_to_ht:nn
730            { \dim_abs:n \l_tmpb_dim }
731            { \vfill \hbox_to_wd:nn { \dim_abs:n \l_tmpa_dim } { } }
732        }
733        { #1 }
734        { }
735      \end { pgfscope }
736    }
```

8   The options

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

```
737  \tl_new:N \l_@@_caption_tl
738  \tl_new:N \l_@@_short_caption_tl
739  \tl_new:N \l_@@_label_tl
```

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

```
740  \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.

```
741  \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.

```
742 \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).

```
743 \dim_new:N \l_@@_cell_space_top_limit_dim
744 \dim_new:N \l_@@_cell_space_bottom_limit_dim
```

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

```
745 \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.

```
746 \dim_new:N \l_@@_xdots_inter_dim
747 \hook_gput_code:nnn { begindocument } { . }
748   { \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).

```
749 \dim_new:N \l_@@_xdots_shorten_start_dim
750 \dim_new:N \l_@@_xdots_shorten_end_dim
751 \hook_gput_code:nnn { begindocument } { . }
752   {
753     \dim_set:Nn \l_@@_xdots_shorten_start_dim { 0.3 em }
754     \dim_set:Nn \l_@@_xdots_shorten_end_dim { 0.3 em }
755   }
```

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.

```
756 \dim_new:N \l_@@_xdots_radius_dim
757 \hook_gput_code:nnn { begindocument } { . }
758   { \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.

```
759 \tl_new:N \l_@@_xdots_line_style_tl
760 \tl_const:Nn \c_@@_standard_tl { standard }
761 \tl_set_eq:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl
```

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

```
762 \bool_new:N \l_@@_light_syntax_bool
763 \bool_new:N \l_@@_light_syntax_expanded_bool
```

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

```
764 \tl_new:N \l_@@_baseline_tl
765 \tl_set:Nn \l_@@_baseline_tl { c }
```

26

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

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

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

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

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

```
768 \bool_new:N \l_@@_parallelize_diags_bool
769 \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`.

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

```
771 \dim_new:N \l_@@_notes_above_space_dim
772 \hook_gput_code:nnn { begindocument } { . }
773   { \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.).

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

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

```
775 \cs_new_protected:Npn \@@_reset_arraystretch:
776   { \cs_set_nopar:Npn \arraystretch { 1 } }
```

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

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

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

```
778 \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.

```
779 \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".

```
780 \bool_new:N \l_@@_medium_nodes_bool
781 \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).

```
782 \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`).

```
783 \dim_new:N \l_@@_left_margin_dim
784 \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`.

```
785 \dim_new:N \l_@@_extra_left_margin_dim
786 \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.

```
787 \tl_new:N \l_@@_end_of_row_tl
788 \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 ":".

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

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

```
790 \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 acheive 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.

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

```
792 \keys_define:nn { nicematrix / xdots }
793   {
794     shorten-start .code:n =
795       \hook_gput_code:nnn { begindocument } { . }
796         { \dim_set:Nn \l_@@_xdots_shorten_start_dim { #1 } } ,
797     shorten-end .code:n =
798       \hook_gput_code:nnn { begindocument } { . }
799         { \dim_set:Nn \l_@@_xdots_shorten_end_dim { #1 } } ,
800     shorten-start .value_required:n = true ,
801     shorten-end .value_required:n = true ,
802     shorten .code:n =
803       \hook_gput_code:nnn { begindocument } { . }
804         {
805           \dim_set:Nn \l_@@_xdots_shorten_start_dim { #1 }
806           \dim_set:Nn \l_@@_xdots_shorten_end_dim { #1 }
807         } ,
808     shorten .value_required:n = true ,
809     horizontal-labels .bool_set:N = \l_@@_xdots_h_labels_bool ,
810     horizontal-labels .default:n = true ,
811     line-style .code:n =
812       {
813         \bool_lazy_or:nnTF
814           { \cs_if_exist_p:N \tikzpicture }
815           { \str_if_eq_p:nn { #1 } { standard } }
816           { \tl_set:Nn \l_@@_xdots_line_style_tl { #1 } }
817           { \@@_error:n { bad~option~for~line-style } }
818       } ,
```

```
819    line-style .value_required:n = true ,
820    color .tl_set:N = \l_@@_xdots_color_tl ,
821    color .value_required:n = true ,
822    radius .code:n =
823      \hook_gput_code:nnn { begindocument } { . }
824        { \dim_set:Nn \l_@@_xdots_radius_dim { #1 } } ,
825    radius .value_required:n = true ,
826    inter .code:n =
827      \hook_gput_code:nnn { begindocument } { . }
828        { \dim_set:Nn \l_@@_xdots_inter_dim { #1 } } ,
829    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 an absent ^{...}.

```
830    down .code:n = \tl_put_right:Nn \l_@@_xdots_down_tl { #1 } ,
831    up .code:n = \tl_put_right:Nn \l_@@_xdots_up_tl { #1 } ,
832    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 catched when \Ddots or \Iddots is used (during the construction of the array and not when we draw the dotted lines).

```
833    draw-first .code:n = \prg_do_nothing: ,
834    unknown .code:n = \@@_error:n { Unknown~key~for~xdots }
835  }


836 \keys_define:nn { nicematrix / rules }
837  {
838    color .tl_set:N = \l_@@_rules_color_tl ,
839    color .value_required:n = true ,
840    width .dim_set:N = \arrayrulewidth ,
841    width .value_required:n = true ,
842    unknown .code:n = \@@_error:n { Unknown~key~for~rules }
843  }
```

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

```
844 \keys_define:nn { nicematrix / Global }
845  {
846    ampersand-in-blocks .bool_set:N = \l_@@_amp_in_blocks_bool ,
847    ampersand-in-blocks .default:n = true ,
848    &-in-blocks .meta:n = ampersand-in-blocks ,
849    no-cell-nodes .code:n =
850      \cs_set_protected:Npn \@@_node_for_cell:
851        { \box_use_drop:N \l_@@_cell_box } ,
852    no-cell-nodes .value_forbidden:n = true ,
853    rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
854    rounded-corners .default:n = 4 pt ,
855    custom-line .code:n = \@@_custom_line:n { #1 } ,
856    rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
857    rules .value_required:n = true ,
858    standard-cline .bool_set:N = \l_@@_standard_cline_bool ,
859    standard-cline .default:n = true ,
860    cell-space-top-limit .dim_set:N = \l_@@_cell_space_top_limit_dim ,
861    cell-space-top-limit .value_required:n = true ,
862    cell-space-bottom-limit .dim_set:N = \l_@@_cell_space_bottom_limit_dim ,
863    cell-space-bottom-limit .value_required:n = true ,
864    cell-space-limits .meta:n =
865      {
866        cell-space-top-limit = #1 ,
867        cell-space-bottom-limit = #1 ,
868      } ,
```

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

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

```
903              { \cs_set_eq:cN { @@ _ #1 } \@@_make_preamble_vlism:n }
904          }
905          { \@@_error:n { One~letter~allowed } }
906      } ,
907    vlines-in-sub-matrix .value_required:n = true ,
908    hvlines .code:n =
909      {
910        \bool_set_true:N \l_@@_hvlines_bool
911        \tl_set_eq:NN \l_@@_vlines_clist \c_@@_all_tl
912        \tl_set_eq:NN \l_@@_hlines_clist \c_@@_all_tl
913      } ,
914    hvlines-except-borders .code:n =
915      {
916        \tl_set_eq:NN \l_@@_vlines_clist \c_@@_all_tl
917        \tl_set_eq:NN \l_@@_hlines_clist \c_@@_all_tl
918        \bool_set_true:N \l_@@_hvlines_bool
919        \bool_set_true:N \l_@@_except_borders_bool
920      } ,
921    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.

```
922    renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
923    renew-dots .value_forbidden:n = true ,
924    nullify-dots .bool_set:N = \l_@@_nullify_dots_bool ,
925    create-medium-nodes .bool_set:N = \l_@@_medium_nodes_bool ,
926    create-large-nodes .bool_set:N = \l_@@_large_nodes_bool ,
```

```
927    create-extra-nodes .meta:n =
928      { create-medium-nodes , create-large-nodes } ,
929    left-margin .dim_set:N = \l_@@_left_margin_dim ,
930    left-margin .default:n = \arraycolsep ,
931    right-margin .dim_set:N = \l_@@_right_margin_dim ,
932    right-margin .default:n = \arraycolsep ,
933    margin .meta:n = { left-margin = #1 , right-margin = #1 } ,
934    margin .default:n = \arraycolsep ,
935    extra-left-margin .dim_set:N = \l_@@_extra_left_margin_dim ,
936    extra-right-margin .dim_set:N = \l_@@_extra_right_margin_dim ,
937    extra-margin .meta:n =
938      { extra-left-margin = #1 , extra-right-margin = #1 } ,
939    extra-margin .value_required:n = true ,
940    respect-arraystretch .code:n =
941      \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
942    respect-arraystretch .value_forbidden:n = true ,
943    pgf-node-code .tl_set:N = \l_@@_pgf_node_code_tl ,
944    pgf-node-code .value_required:n = true
945  }
```

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

```
946  \keys_define:nn { nicematrix / environments }
947    {
948      corners .clist_set:N = \l_@@_corners_clist ,
949      corners .default:n = { NW , SW , NE , SE } ,
950      code-before .code:n =
951        {
952          \tl_if_empty:nF { #1 }
953            {
954              \tl_gput_left:Nn \g_@@_pre_code_before_tl { #1 }
955              \bool_set_true:N \l_@@_code_before_bool
956            }
957        } ,
958      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}.

```
959      c .code:n = \tl_set:Nn \l_@@_baseline_tl c ,
960      t .code:n = \tl_set:Nn \l_@@_baseline_tl t ,
961      b .code:n = \tl_set:Nn \l_@@_baseline_tl b ,
962      baseline .tl_set:N = \l_@@_baseline_tl ,
963      baseline .value_required:n = true ,
964      columns-width .code:n =
965        \tl_if_eq:nnTF { #1 } { auto }
966          { \bool_set_true:N \l_@@_auto_columns_width_bool }
967          { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,
968      columns-width .value_required:n = true ,
969      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.

```
970        \legacy_if:nF { measuring@ }
971          {
972            \str_set:Ne \l_tmpa_str { #1 }
973            \seq_if_in:NVTF \g_@@_names_seq \l_tmpa_str
974              { \@@_error:nn { Duplicate~name } { #1 } }
975              { \seq_gput_left:NV \g_@@_names_seq \l_tmpa_str }
976            \str_set_eq:NN \l_@@_name_str \l_tmpa_str
977          } ,
978      name .value_required:n = true ,
979      code-after .tl_gset:N = \g_nicematrix_code_after_tl ,
```

```
980    code-after .value_required:n = true ,
981    color-inside .code:n =
982      \bool_set_true:N \l_@@_color_inside_bool
983      \bool_set_true:N \l_@@_code_before_bool ,
984    color-inside .value_forbidden:n = true ,
985    colortbl-like .meta:n = color-inside
986  }
987 \keys_define:nn { nicematrix / notes }
988  {
989    para .bool_set:N = \l_@@_notes_para_bool ,
990    para .default:n = true ,
991    code-before .tl_set:N = \l_@@_notes_code_before_tl ,
992    code-before .value_required:n = true ,
993    code-after .tl_set:N = \l_@@_notes_code_after_tl ,
994    code-after .value_required:n = true ,
995    bottomrule .bool_set:N = \l_@@_notes_bottomrule_bool ,
996    bottomrule .default:n = true ,
997    style .cs_set:Np = \@@_notes_style:n #1 ,
998    style .value_required:n = true ,
999    label-in-tabular .cs_set:Np = \@@_notes_label_in_tabular:n #1 ,
1000   label-in-tabular .value_required:n = true ,
1001   label-in-list .cs_set:Np = \@@_notes_label_in_list:n #1 ,
1002   label-in-list .value_required:n = true ,
1003   enumitem-keys .code:n =
1004     {
1005       \hook_gput_code:nnn { begindocument } { . }
1006         {
1007           \IfPackageLoadedTF { enumitem }
1008             { \setlist* [ tabularnotes ] { #1 } }
1009             { }
1010         }
1011     } ,
1012   enumitem-keys .value_required:n = true ,
1013   enumitem-keys-para .code:n =
1014     {
1015       \hook_gput_code:nnn { begindocument } { . }
1016         {
1017           \IfPackageLoadedTF { enumitem }
1018             { \setlist* [ tabularnotes* ] { #1 } }
1019             { }
1020         }
1021     } ,
1022   enumitem-keys-para .value_required:n = true ,
1023   detect-duplicates .bool_set:N = \l_@@_notes_detect_duplicates_bool ,
1024   detect-duplicates .default:n = true ,
1025   unknown .code:n  = \@@_error:n { Unknown~key~for~notes }
1026  }
1027 \keys_define:nn { nicematrix / delimiters }
1028  {
1029    max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1030    max-width .default:n = true ,
1031    color .tl_set:N = \l_@@_delimiters_color_tl ,
1032    color .value_required:n = true ,
1033  }
```

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

```
1034 \keys_define:nn { nicematrix }
1035  {
1036    NiceMatrixOptions .inherit:n =
1037      { nicematrix / Global } ,
1038    NiceMatrixOptions / xdots .inherit:n = nicematrix / xdots ,
```

```
1039    NiceMatrixOptions / rules .inherit:n = nicematrix / rules ,
1040    NiceMatrixOptions / notes .inherit:n = nicematrix / notes ,
1041    NiceMatrixOptions / sub-matrix .inherit:n = nicematrix / sub-matrix ,
1042    SubMatrix / rules .inherit:n = nicematrix / rules ,
1043    CodeAfter / xdots .inherit:n = nicematrix / xdots ,
1044    CodeBefore / sub-matrix .inherit:n = nicematrix / sub-matrix ,
1045    CodeAfter / sub-matrix .inherit:n = nicematrix / sub-matrix ,
1046    NiceMatrix .inherit:n =
1047      {
1048        nicematrix / Global ,
1049        nicematrix / environments ,
1050      } ,
1051    NiceMatrix / xdots .inherit:n = nicematrix / xdots ,
1052    NiceMatrix / rules .inherit:n = nicematrix / rules ,
1053    NiceTabular .inherit:n =
1054      {
1055        nicematrix / Global ,
1056        nicematrix / environments
1057      } ,
1058    NiceTabular / xdots .inherit:n = nicematrix / xdots ,
1059    NiceTabular / rules .inherit:n = nicematrix / rules ,
1060    NiceTabular / notes .inherit:n = nicematrix / notes ,
1061    NiceArray .inherit:n =
1062      {
1063        nicematrix / Global ,
1064        nicematrix / environments ,
1065      } ,
1066    NiceArray / xdots .inherit:n = nicematrix / xdots ,
1067    NiceArray / rules .inherit:n = nicematrix / rules ,
1068    pNiceArray .inherit:n =
1069      {
1070        nicematrix / Global ,
1071        nicematrix / environments ,
1072      } ,
1073    pNiceArray / xdots .inherit:n = nicematrix / xdots ,
1074    pNiceArray / rules .inherit:n = nicematrix / rules ,
1075  }
```

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

```
1076  \keys_define:nn { nicematrix / NiceMatrixOptions }
1077    {
1078      delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1079      delimiters / color .value_required:n = true ,
1080      delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1081      delimiters / max-width .default:n = true ,
1082      delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
1083      delimiters .value_required:n = true ,
1084      width .dim_set:N = \l_@@_width_dim ,
1085      width .value_required:n = true ,
1086      last-col .code:n =
1087        \tl_if_empty:nF { #1 }
1088          { \@@_error:n { last-col~non~empty~for~NiceMatrixOptions } }
1089          \int_zero:N \l_@@_last_col_int ,
1090      small .bool_set:N = \l_@@_small_bool ,
1091      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.

```
1092      renew-matrix .code:n = \@@_renew_matrix: ,
1093      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}`.

```
1094       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.

```
1095       columns-width .code:n =
1096         \tl_if_eq:nnTF { #1 } { auto }
1097           { \@@_error:n { Option~auto~for~columns-width } }
1098           { \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.

```
1099       allow-duplicate-names .code:n =
1100         \@@_msg_redirect_name:nn { Duplicate~name } { none } ,
1101       allow-duplicate-names .value_forbidden:n = true ,
1102       notes .code:n = \keys_set:nn { nicematrix / notes } { #1 } ,
1103       notes .value_required:n = true ,
1104       sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1105       sub-matrix .value_required:n = true ,
1106       matrix / columns-type .tl_set:N = \l_@@_columns_type_tl ,
1107       matrix / columns-type .value_required:n = true ,
1108       caption-above .bool_set:N = \l_@@_caption_above_bool ,
1109       caption-above .default:n = true ,
1110       unknown .code:n  = \@@_error:n { Unknown~key~for~NiceMatrixOptions }
1111     }
```

`\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.

```
1112 \NewDocumentCommand \NiceMatrixOptions { m }
1113   { \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.

```
1114 \keys_define:nn { nicematrix / NiceMatrix }
1115   {
1116     last-col .code:n = \tl_if_empty:nTF { #1 }
1117                       {
1118                         \bool_set_true:N \l_@@_last_col_without_value_bool
1119                         \int_set:Nn \l_@@_last_col_int { -1 }
1120                       }
1121                       { \int_set:Nn \l_@@_last_col_int { #1 } } ,
1122     columns-type .tl_set:N = \l_@@_columns_type_tl ,
1123     columns-type .value_required:n = true ,
1124     l .meta:n = { columns-type = l } ,
1125     r .meta:n = { columns-type = r } ,
1126     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1127     delimiters / color .value_required:n = true ,
1128     delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1129     delimiters / max-width .default:n = true ,
1130     delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
1131     delimiters .value_required:n = true ,
1132     small .bool_set:N = \l_@@_small_bool ,
1133     small .value_forbidden:n = true ,
1134     unknown .code:n = \@@_error:n { Unknown~key~for~NiceMatrix }
1135   }
```

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

```
1136 \keys_define:nn { nicematrix / NiceArray }
1137   {
```

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.

```
1138     small .bool_set:N = \l_@@_small_bool ,
1139     small .value_forbidden:n = true ,
1140     last-col .code:n = \tl_if_empty:nF { #1 }
1141                        { \@@_error:n { last-col~non~empty~for~NiceArray } } }
1142                        \int_zero:N \l_@@_last_col_int ,
1143     r .code:n = \@@_error:n { r~or~l~with~preamble } ,
1144     l .code:n = \@@_error:n { r~or~l~with~preamble } ,
1145     unknown .code:n = \@@_error:n { Unknown~key~for~NiceArray }
1146   }
1147 \keys_define:nn { nicematrix / pNiceArray }
1148   {
1149     first-col .code:n = \int_zero:N \l_@@_first_col_int ,
1150     last-col .code:n = \tl_if_empty:nF {#1}
1151                        { \@@_error:n { last-col~non~empty~for~NiceArray } } }
1152                        \int_zero:N \l_@@_last_col_int ,
1153     first-row .code:n = \int_zero:N \l_@@_first_row_int ,
1154     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1155     delimiters / color .value_required:n = true ,
1156     delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1157     delimiters / max-width .default:n = true ,
1158     delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
1159     delimiters .value_required:n = true ,
1160     small .bool_set:N = \l_@@_small_bool ,
1161     small .value_forbidden:n = true ,
1162     r .code:n = \@@_error:n { r~or~l~with~preamble } ,
1163     l .code:n = \@@_error:n { r~or~l~with~preamble } ,
1164     unknown .code:n = \@@_error:n { Unknown~key~for~NiceMatrix }
1165   }
```

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

```
1166 \keys_define:nn { nicematrix / NiceTabular }
1167   {
```

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.

```
1168     width .code:n = \dim_set:Nn \l_@@_width_dim { #1 }
1169                     \bool_set_true:N \l_@@_width_used_bool ,
1170     width .value_required:n = true ,
1171     notes .code:n = \keys_set:nn { nicematrix / notes } { #1 } ,
1172     tabularnote .tl_gset:N = \g_@@_tabularnote_tl ,
1173     tabularnote .value_required:n = true ,
1174     caption .tl_set:N = \l_@@_caption_tl ,
1175     caption .value_required:n = true ,
1176     short-caption .tl_set:N = \l_@@_short_caption_tl ,
1177     short-caption .value_required:n = true ,
1178     label .tl_set:N = \l_@@_label_tl ,
1179     label .value_required:n = true ,
1180     last-col .code:n = \tl_if_empty:nF {#1}
1181                        { \@@_error:n { last-col~non~empty~for~NiceArray } } }
1182                        \int_zero:N \l_@@_last_col_int ,
1183     r .code:n = \@@_error:n { r~or~l~with~preamble } ,
1184     l .code:n = \@@_error:n { r~or~l~with~preamble } ,
1185     unknown .code:n = \@@_error:n { Unknown~key~for~NiceTabular }
1186   }
```

35

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
```

```
1187 \keys_define:nn { nicematrix / CodeAfter }
1188   {
1189     delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1190     delimiters / color .value_required:n = true ,
1191     rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
1192     rules .value_required:n = true ,
1193     xdots .code:n = \keys_set:nn { nicematrix / xdots } { #1 } ,
1194     sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1195     sub-matrix .value_required:n = true ,
1196     unknown .code:n = \@@_error:n { Unknown~key~for~CodeAfter }
1197   }
```

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}).

```
1198 \cs_new_protected:Npn \@@_cell_begin:w
1199   {
```

\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.

```
1200     \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).

```
1201     \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
```

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

```
1202     \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.

```
1203     \int_compare:nNnT \c@jCol = \c_one_int
1204       { \int_compare:nNnT \l_@@_first_col_int = \c_one_int \@@_begin_of_row: }
```

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

```
1205     \hbox_set:Nw \l_@@_cell_box
```

The following command is nullified in the tabulars.

```
1206     \@@_tuning_not_tabular_begin:
```

```
1207     \@@_tuning_first_row:
1208     \@@_tuning_last_row:
1209     \g_@@_row_style_tl
1210   }
```

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

```
1211 \cs_new_protected:Npn \@@_tuning_first_row:
1212   {
1213     \int_if_zero:nT \c@iRow
1214       {
1215         \int_compare:nNnT \c@jCol > \c_zero_int
1216           {
1217             \l_@@_code_for_first_row_tl
1218             \xglobal \colorlet { nicematrix-first-row } { . }
1219           }
1220       }
1221   }
```

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

```
1222 \cs_new_protected:Npn \@@_tuning_last_row:
1223   {
1224     \int_compare:nNnT \c@iRow = \l_@@_last_row_int
1225       {
1226         \l_@@_code_for_last_row_tl
1227         \xglobal \colorlet { nicematrix-last-row } { . }
1228       }
1229   }
```

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

```
1230 \cs_set_eq:NN \@@_tuning_key_small: \prg_do_nothing:
```

The following commands are nullified in the tabulars.

```
1231 \cs_set_nopar:Npn \@@_tuning_not_tabular_begin:
1232   {
1233     \c_math_toggle_token
```

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

```
1234     \@@_tuning_key_small:
1235   }
1236 \cs_set_eq:NN \@@_tuning_not_tabular_end: \c_math_toggle_token
```

The following macro `\@@_begin_of_row` is usually used in the cell number 1 of the row. However, when the key `first-col` is used, `\@@_begin_of_row` is executed in the cell number 0 of the row.

```
1237 \cs_new_protected:Npn \@@_begin_of_row:
1238   {
1239     \int_gincr:N \c@iRow
1240     \dim_gset_eq:NN \g_@@_dp_ante_last_row_dim \g_@@_dp_last_row_dim
1241     \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
1242     \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1243     \pgfpicture
1244     \pgfrememberpicturepositiononpagetrue
1245     \pgfcoordinate
1246       { \@@_env: - row - \int_use:N \c@iRow - base }
1247       { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
1248     \str_if_empty:NF \l_@@_name_str
1249       {
1250         \pgfnodealias
1251           { \l_@@_name_str - row - \int_use:N \c@iRow - base }
1252           { \@@_env: - row - \int_use:N \c@iRow - base }
1253       }
1254     \endpgfpicture
1255   }
```

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.

```
1256 \cs_new_protected:Npn \@@_update_for_first_and_last_row:
1257   {
1258     \int_if_zero:nTF \c@iRow
1259       {
1260         \dim_gset:Nn \g_@@_dp_row_zero_dim
1261           { \dim_max:nn \g_@@_dp_row_zero_dim { \box_dp:N \l_@@_cell_box } }
1262         \dim_gset:Nn \g_@@_ht_row_zero_dim
1263           { \dim_max:nn \g_@@_ht_row_zero_dim { \box_ht:N \l_@@_cell_box } }
1264       }
1265       {
1266         \int_compare:nNnT \c@iRow = \c_one_int
1267           {
1268             \dim_gset:Nn \g_@@_ht_row_one_dim
1269               { \dim_max:nn \g_@@_ht_row_one_dim { \box_ht:N \l_@@_cell_box } }
1270           }
1271       }
1272   }
1273 \cs_new_protected:Npn \@@_rotate_cell_box:
1274   {
1275     \box_rotate:Nn \l_@@_cell_box { 90 }
1276     \bool_if:NTF \g_@@_rotate_c_bool
1277       {
1278         \hbox_set:Nn \l_@@_cell_box
1279           {
1280             \c_math_toggle_token
1281             \vcenter { \box_use:N \l_@@_cell_box }
1282             \c_math_toggle_token
1283           }
1284       }
1285       {
1286         \int_compare:nNnT \c@iRow = \l_@@_last_row_int
1287           {
1288             \vbox_set_top:Nn \l_@@_cell_box
1289               {
1290                 \vbox_to_zero:n { }
1291                 \skip_vertical:n { - \box_ht:N \@arstrutbox + 0.8 ex }
1292                 \box_use:N \l_@@_cell_box
1293               }
1294           }
1295       }
1296     \bool_gset_false:N \g_@@_rotate_bool
1297     \bool_gset_false:N \g_@@_rotate_c_bool
1298   }
1299 \cs_new_protected:Npn \@@_adjust_size_box:
1300   {
1301     \dim_compare:nNnT \g_@@_blocks_wd_dim > \c_zero_dim
1302       {
1303         \box_set_wd:Nn \l_@@_cell_box
1304           { \dim_max:nn { \box_wd:N \l_@@_cell_box } \g_@@_blocks_wd_dim }
1305         \dim_gzero:N \g_@@_blocks_wd_dim
1306       }
1307     \dim_compare:nNnT \g_@@_blocks_dp_dim > \c_zero_dim
1308       {
1309         \box_set_dp:Nn \l_@@_cell_box
1310           { \dim_max:nn { \box_dp:N \l_@@_cell_box } \g_@@_blocks_dp_dim }
1311         \dim_gzero:N \g_@@_blocks_dp_dim
1312       }
1313     \dim_compare:nNnT \g_@@_blocks_ht_dim > \c_zero_dim
1314       {
```

```
1315        \box_set_ht:Nn \l_@@_cell_box
1316          { \dim_max:nn { \box_ht:N \l_@@_cell_box } \g_@@_blocks_ht_dim }
1317        \dim_gzero:N \g_@@_blocks_ht_dim
1318      }
1319   }
1320 \cs_new_protected:Npn \@@_cell_end:
1321   {
```

The following command is nullified in the tabulars.

```
1322      \@@_tuning_not_tabular_end:
1323      \hbox_set_end:
1324      \@@_cell_end_i:
1325   }
1326 \cs_new_protected:Npn \@@_cell_end_i:
1327   {
```

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.

```
1328      \g_@@_cell_after_hook_tl
1329      \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
1330      \@@_adjust_size_box:
1331      \box_set_ht:Nn \l_@@_cell_box
1332        { \box_ht:N \l_@@_cell_box + \l_@@_cell_space_top_limit_dim }
1333      \box_set_dp:Nn \l_@@_cell_box
1334        { \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").

```
1335      \@@_update_max_cell_width:
```

The following computations are for the "first row" and the "last row".

```
1336      \@@_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.

```
1337      \bool_if:NTF \g_@@_empty_cell_bool
1338        { \box_use_drop:N \l_@@_cell_box }
1339        {
1340          \bool_if:NTF \g_@@_not_empty_cell_bool
1341            \@@_node_for_cell:
1342            {
1343              \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > \c_zero_dim
1344                \@@_node_for_cell:
```

```
1345            { \box_use_drop:N \l_@@_cell_box }
1346         }
1347      }
1348    \int_gset:Nn \g_@@_col_total_int { \int_max:nn \g_@@_col_total_int \c@jCol }
1349    \bool_gset_false:N \g_@@_empty_cell_bool
1350    \bool_gset_false:N \g_@@_not_empty_cell_bool
1351  }
```

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

```
1352 \cs_new_protected:Npn \@@_update_max_cell_width:
1353  {
1354    \dim_gset:Nn \g_@@_max_cell_width_dim
1355      { \dim_max:nn \g_@@_max_cell_width_dim { \box_wd:N \l_@@_cell_box } } }
1356  }
```

The following variant of `\@@_cell_end:` is only for the columns of type `w{s}{...}` or `W{s}{...}` (which use the horizontal alignement key `s` of `\makebox`).

```
1357 \cs_new_protected:Npn \@@_cell_end_for_w_s:
1358  {
1359    \@@_math_toggle:
1360    \hbox_set_end:
1361    \bool_if:NF \g_@@_rotate_bool
1362      {
1363        \hbox_set:Nn \l_@@_cell_box
1364          {
1365            \makebox [ \l_@@_col_width_dim ] [ s ]
1366              { \hbox_unpack_drop:N \l_@@_cell_box }
1367          }
1368      }
1369    \@@_cell_end_i:
1370  }
```

```
1371 \pgfset
1372  {
1373    nicematrix / cell-node /.style =
1374      {
1375        inner~sep = \c_zero_dim ,
1376        minimum~width = \c_zero_dim
1377      }
1378  }
```

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

```
1379 \cs_new_protected:Npn \@@_node_for_cell:
1380  {
1381    \pgfpicture
1382    \pgfsetbaseline \c_zero_dim
1383    \pgfrememberpicturepositiononpagetrue
1384    \pgfset { nicematrix / cell-node }
1385    \pgfnode
1386      { rectangle }
1387      { base }
1388      {
```

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

```
1389        \set@color
1390        \box_use_drop:N \l_@@_cell_box
1391      }
1392      { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1393      { \l_@@_pgf_node_code_tl }
```

```
1394    \str_if_empty:NF \l_@@_name_str
1395      {
1396        \pgfnodealias
1397          { \l_@@_name_str - \int_use:N \c@iRow - \int_use:N \c@jCol }
1398          { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1399      }
1400    \endpgfpicture
1401  }
```

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.

```
1402 \cs_new_protected:Npn \@@_patch_node_for_cell:n #1
1403   {
1404     \cs_new_protected:Npn \@@_patch_node_for_cell:
1405       {
1406         \hbox_set:Nn \l_@@_cell_box
1407           {
1408             \box_move_up:nn { \box_ht:N \l_@@_cell_box}
1409             \hbox_overlap_left:n
1410               {
1411                 \pgfsys@markposition
1412                   { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - NW }
```

I don't know why the following adjustement 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.

```
1413                 #1
1414               }
1415             \box_use:N \l_@@_cell_box
1416             \box_move_down:nn { \box_dp:N \l_@@_cell_box }
1417             \hbox_overlap_left:n
1418               {
1419                 \pgfsys@markposition
1420                   { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - SE }
1421                 #1
1422               }
1423           }
1424       }
1425   }
```

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

```
1426 \bool_lazy_or:nnTF \sys_if_engine_xetex_p: \sys_if_output_dvi_p:
1427   {
1428     \@@_patch_node_for_cell:n
1429       { \skip_horizontal:n { 0.5 \box_wd:N \l_@@_cell_box } }
1430   }
1431   { \@@_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}
```
$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 \cdots\cdots\cdots\cdots 6 \\ 7 \cdots\cdots\cdots\cdots \end{pmatrix}$$

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

`\@@_draw_Cdots:nnn {2}{2}{}`

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

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

```
1432 \cs_new_protected:Npn \@@_instruction_of_type:nnn #1 #2 #3
1433   {
1434     \bool_if:nTF { #1 } \tl_gput_left:cx \tl_gput_right:cx
1435       { g_@@_ #2 _ lines _ tl }
1436       {
1437         \use:c { @@ _ draw _ #2 : nnn }
1438           { \int_use:N \c@iRow }
1439           { \int_use:N \c@jCol }
1440           { \exp_not:n { #3 } } }
1441       }
1442   }
```

```
1443 \cs_new_protected:Npn \@@_array:
1444   {
1445 %     \begin{macrocode}
1446     \dim_set:Nn \col@sep
1447       { \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
1448     \dim_compare:nNnTF \l_@@_tabular_width_dim = \c_zero_dim
1449       { \cs_set_nopar:Npn \@halignto { } }
1450       { \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.

```
1451     \@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:onTF is fully expandable and we need something fully expandable here.

```
1452     [ \str_if_eq:onTF \l_@@_baseline_tl c c t ]
1453   }
```

We keep in memory the standard version of \ialign because we will redefine \ialign in the environment {NiceArrayWithDelims} but restore the standard version for use in the cells of the array. However, since version 2.6a (version for the Tagging Project), array uses \ar@ialign instead of \ialign. In that case, of course, you do a saving of \ar@ialign.

```
1454 \bool_if:NTF \c_@@_tagging_array_bool
1455   { \cs_set_eq:NN \@@_old_ar@ialign: \ar@ialign }
1456   { \cs_set_eq:NN \@@_old_ialign: \ialign }
```

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

```
1457 \cs_new_protected:Npn \@@_create_row_node:
1458   {
1459     \int_compare:nNnT \c@iRow > \g_@@_last_row_node_int
1460       {
1461         \int_gset_eq:NN \g_@@_last_row_node_int \c@iRow
1462         \@@_create_row_node_i:
1463       }
1464   }
```

```
1465 \cs_new_protected:Npn \@@_create_row_node_i:
1466   {
```

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

```
1467     \hbox
1468       {
1469         \bool_if:NT \l_@@_code_before_bool
1470           {
1471             \vtop
1472               {
```

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

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

```
1492 \cs_new:Npn \@@_everycr: { \noalign { \@@_everycr_i: } }
```

```
1493 \cs_new_protected:Npn \@@_everycr_i:
1494    {
1495      \bool_if:NT \c_@@_testphase_table_bool
1496        {
1497          \tbl_if_row_was_started:T { \UseTaggingSocket { tbl / row / end } }
1498          \tbl_update_cell_data_for_next_row:
1499        }
1500      \int_gzero:N \c@jCol
1501      \bool_gset_false:N \g_@@_after_col_zero_bool
1502      \bool_if:NF \g_@@_row_of_col_done_bool
1503        {
1504          \@@_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).

```
1505          \tl_if_empty:NF \l_@@_hlines_clist
1506            {
1507              \tl_if_eq:NNF \l_@@_hlines_clist \c_@@_all_tl
1508                {
1509                  \exp_args:NNe
1510                    \clist_if_in:NnT
1511                    \l_@@_hlines_clist
1512                    { \int_eval:n { \c@iRow + 1 } }
1513                }
1514                {
```

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.

```
1515                  \int_compare:nNnT \c@iRow > { -1 }
1516                    {
1517                      \int_compare:nNnF \c@iRow = \l_@@_last_row_int
1518                        { \hrule height \arrayrulewidth width \c_zero_dim }
1519                    }
1520                }
1521            }
1522        }
1523    }
```

43

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

```
1524 \cs_set_protected:Npn \@@_renew_dots:
1525   {
1526     \cs_set_eq:NN \ldots \@@_Ldots
1527     \cs_set_eq:NN \cdots \@@_Cdots
1528     \cs_set_eq:NN \vdots \@@_Vdots
1529     \cs_set_eq:NN \ddots \@@_Ddots
1530     \cs_set_eq:NN \iddots \@@_Iddots
1531     \cs_set_eq:NN \dots \@@_Ldots
1532     \cs_set_eq:NN \hdotsfor \@@_Hdotsfor:
1533   }

1534 \cs_new_protected:Npn \@@_test_color_inside:
1535   {
1536     \bool_if:NF \l_@@_color_inside_bool
1537       {
```

We will issue an error only during the first run.

```
1538         \bool_if:NF \g_@@_aux_found_bool
1539           { \@@_error:n { without~color-inside } }
1540       }
1541   }
```

```
1542 \cs_new_protected:Npn \@@_redefine_everycr: { \everycr { \@@_everycr: } }
1543   \hook_gput_code:nnn { begindocument } { . }
1544     {
1545       \IfPackageLoadedTF { colortbl }
1546         {
1547           \cs_set_protected:Npn \@@_redefine_everycr:
1548             {
1549               \CT@everycr
1550                 {
1551                   \noalign { \cs_gset_eq:NN \CT@row@color \prg_do_nothing: }
1552                   \@@_everycr:
1553                 }
1554             }
1555         }
1556         { }
1557     }
```

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].

```
1558 \hook_gput_code:nnn { begindocument } { . }
1559   {
1560     \IfPackageLoadedTF { booktabs }
1561       {
1562         \cs_new_protected:Npn \@@_patch_booktabs:
1563           { \tl_put_left:Nn \@BTnormal \@@_create_row_node_i: }
1564       }
1565       { \cs_new_protected:Npn \@@_patch_booktabs: { } }
1566   }
```

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`

---

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

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

(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.

```
1567 \cs_new_protected:Npn \@@_some_initialization:
1568   {
1569     \dim_gset:Nn \g_@@_dp_row_zero_dim { \box_dp:N \@arstrutbox }
1570     \dim_gset:Nn \g_@@_ht_row_zero_dim { \box_ht:N \@arstrutbox }
1571     \dim_gset_eq:NN \g_@@_ht_row_one_dim \g_@@_ht_row_zero_dim
1572     \dim_gzero:N \g_@@_dp_ante_last_row_dim
1573     \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1574     \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
1575   }
```

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

```
1576 \cs_new_protected:Npn \@@_pre_array_ii:
1577   {
```

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

```
1578     \int_gzero:N \g_@@_total_X_weight_int

1579     \@@_expand_clist:N \l_@@_hlines_clist
1580     \@@_expand_clist:N \l_@@_vlines_clist
1581     \@@_patch_booktabs:
1582     \box_clear_new:N \l_@@_cell_box
1583     \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}).

```
1584     \bool_if:NT \l_@@_small_bool
1585       {
1586         \cs_set_nopar:Npn \arraystretch { 0.47 }
1587         \dim_set:Nn \arraycolsep { 1.45 pt }
```

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

```
1588         \cs_set_eq:NN \@@_tuning_key_small: \scriptstyle
1589       }


1590     \bool_if:NT \g_@@_recreate_cell_nodes_bool
1591       {
1592         \tl_put_right:Nn \@@_begin_of_row:
1593           {
1594             \pgfsys@markposition
1595               { \@@_env: - row - \int_use:N \c@iRow - base }
1596           }
1597       }
```

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.

```
1598     \bool_if:NTF \c_@@_tagging_array_bool
1599       {
1600         \cs_set_nopar:Npn \ar@ialign
1601           {
1602             \bool_if:NT \c_@@_testphase_table_bool \tbl_init_cell_data_for_table:
1603             \@@_redefine_everycr:
1604             \dim_zero:N \tabskip
1605             \@@_some_initialization:
```

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

```
1606            \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign:
1607            \halign
1608          }
1609        }
```

The following part will be deleted when we will delete the boolean `\c_@@_tagging_array_bool` (when we consider the version 2.6a of array is required).

```
1610        {
1611          \cs_set_nopar:Npn \ialign
1612            {
1613              \@@_redefine_everycr:
1614              \dim_zero:N \tabskip
1615              \@@_some_initialization:
1616              \cs_set_eq:NN \ialign \@@_old_ialign:
1617              \halign
1618            }
1619        }
```

We keep in memory the old versions or `\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).

```
1620        \cs_set_eq:NN \@@_old_ldots \ldots
1621        \cs_set_eq:NN \@@_old_cdots \cdots
1622        \cs_set_eq:NN \@@_old_vdots \vdots
1623        \cs_set_eq:NN \@@_old_ddots \ddots
1624        \cs_set_eq:NN \@@_old_iddots \iddots
1625        \bool_if:NTF \l_@@_standard_cline_bool
1626          { \cs_set_eq:NN \cline \@@_standard_cline }
1627          { \cs_set_eq:NN \cline \@@_cline }
1628        \cs_set_eq:NN \Ldots \@@_Ldots
1629        \cs_set_eq:NN \Cdots \@@_Cdots
1630        \cs_set_eq:NN \Vdots \@@_Vdots
1631        \cs_set_eq:NN \Ddots \@@_Ddots
1632        \cs_set_eq:NN \Iddots \@@_Iddots
1633        \cs_set_eq:NN \Hline \@@_Hline:
1634        \cs_set_eq:NN \Hspace \@@_Hspace:
1635        \cs_set_eq:NN \Hdotsfor \@@_Hdotsfor:
1636        \cs_set_eq:NN \Vdotsfor \@@_Vdotsfor:
1637        \cs_set_eq:NN \Block \@@_Block:
1638        \cs_set_eq:NN \rotate \@@_rotate:
1639        \cs_set_eq:NN \OnlyMainNiceMatrix \@@_OnlyMainNiceMatrix:n
1640        \cs_set_eq:NN \dotfill \@@_dotfill:
1641        \cs_set_eq:NN \CodeAfter \@@_CodeAfter:
1642        \cs_set_eq:NN \diagbox \@@_diagbox:nn
1643        \cs_set_eq:NN \NotEmpty \@@_NotEmpty:
1644        \cs_set_eq:NN \RowStyle \@@_RowStyle:n
1645        \seq_map_inline:Nn \l_@@_custom_line_commands_seq
1646          { \cs_set_eq:cc { ##1 } { nicematrix - ##1 } }
1647        \cs_set_eq:NN \cellcolor \@@_cellcolor_tabular
1648        \cs_set_eq:NN \rowcolor \@@_rowcolor_tabular
1649        \cs_set_eq:NN \rowcolors \@@_rowcolors_tabular
1650        \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors_tabular
1651        \int_compare:nNnT \l_@@_first_row_int > \c_zero_int
1652          { \cs_set_eq:NN \@@_tuning_first_row: \prg_do_nothing: }
1653        \int_compare:nNnT \l_@@_last_row_int < \c_zero_int
1654          { \cs_set_eq:NN \@@_tuning_last_row: \prg_do_nothing: }
1655        \bool_if:NT \l_@@_renew_dots_bool \@@_renew_dots:
```

We redefine `\multicolumn` and, since we want `\multicolumn` to be available in the potential environments {tabular} nested in the environments of nicematrix, we patch {tabular} to go back to the original definition. A `\hook_gremove_code:nn` will be put in `\@@_after_array:`.

```
1656        \cs_set_eq:NN \multicolumn \@@_multicolumn:nnn
```

```
1657        \hook_gput_code:nnn { env / tabular / begin } { nicematrix }
1658          { \cs_set_eq:NN \multicolumn \@@_old_multicolumn }
1659        \@@_revert_colortbl:
```

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

```
1660        \tl_if_exist:NT \l_@@_note_in_caption_tl
1661          {
1662            \tl_if_empty:NF \l_@@_note_in_caption_tl
1663              {
1664                \int_gset_eq:NN \g_@@_notes_caption_int \l_@@_note_in_caption_tl
1665                \int_gset:Nn \c@tabularnote { \l_@@_note_in_caption_tl }
1666              }
1667          }
```

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).

```
1668        \seq_gclear:N \g_@@_multicolumn_cells_seq
1669        \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).

```
1670        \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).

```
1671        \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.

```
1672        \int_gzero_new:N \g_@@_col_total_int

1673        \cs_set_eq:NN \@ifnextchar \new@ifnextchar

1674        \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.

```
1675        \tl_gclear_new:N \g_@@_Cdots_lines_tl
1676        \tl_gclear_new:N \g_@@_Ldots_lines_tl
1677        \tl_gclear_new:N \g_@@_Vdots_lines_tl
1678        \tl_gclear_new:N \g_@@_Ddots_lines_tl
1679        \tl_gclear_new:N \g_@@_Iddots_lines_tl
1680        \tl_gclear_new:N \g_@@_HVdotsfor_lines_tl

1681        \tl_gclear:N \g_nicematrix_code_before_tl
1682        \tl_gclear:N \g_@@_pre_code_before_tl
1683      }
```

This is the end of \@@_pre_array_ii:.

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

```
1684  \cs_new_protected:Npn \@@_pre_array:
1685    {
1686      \cs_if_exist:NT \theiRow { \int_set_eq:NN \l_@@_old_iRow_int \c@iRow }
1687      \int_gzero_new:N \c@iRow
1688      \cs_if_exist:NT \thejCol { \int_set_eq:NN \l_@@_old_jCol_int \c@jCol }
1689      \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).

```
1690        \int_compare:nNnT \l_@@_last_row_int = { -1 }
1691          {
1692            \bool_set_true:N \l_@@_last_row_without_value_bool
1693            \bool_if:NT \g_@@_aux_found_bool
1694              { \int_set:Nn \l_@@_last_row_int { \seq_item:Nn \g_@@_size_seq 3 } }
1695          }
1696        \int_compare:nNnT \l_@@_last_col_int = { -1 }
1697          {
1698            \bool_if:NT \g_@@_aux_found_bool
1699              { \int_set:Nn \l_@@_last_col_int { \seq_item:Nn \g_@@_size_seq 6 } }
1700          }
```

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".

```
1701        \int_compare:nNnT \l_@@_last_row_int > { -2 }
1702          {
1703            \tl_put_right:Nn \@@_update_for_first_and_last_row:
1704              {
1705                \dim_gset:Nn \g_@@_ht_last_row_dim
1706                  { \dim_max:nn \g_@@_ht_last_row_dim { \box_ht:N \l_@@_cell_box } }
1707                \dim_gset:Nn \g_@@_dp_last_row_dim
1708                  { \dim_max:nn \g_@@_dp_last_row_dim { \box_dp:N \l_@@_cell_box } }
1709              }
1710          }
```

```
1711        \seq_gclear:N \g_@@_cols_vlism_seq
1712        \seq_gclear:N \g_@@_submatrix_seq
```

Now the `\CodeBefore`.

```
1713        \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.

```
1714        \seq_gclear:N \g_@@_pos_of_blocks_seq
```

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

```
1715        \seq_gclear_new:N \g_@@_multicolumn_cells_seq
1716        \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 interfers with the construction of the last row-node of the array. We don't want to create such row-node twice (to avaid warnings or, maybe, errors). That's why the command `\@@_create_row_node:` will use the following counter to avoid such construction.

```
1717        \int_gset:Nn \g_@@_last_row_node_int { -2 }
```

The value −2 is important.

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

```
1718        \@@_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.

```
1719        \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]).

```
1720        \dim_zero_new:N \l_@@_left_delim_dim
1721        \dim_zero_new:N \l_@@_right_delim_dim
1722        \bool_if:NTF \g_@@_delims_bool
1723          {
```

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

```
1724            \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_left_delim_tl $ }
1725            \dim_set:Nn \l_@@_left_delim_dim { \box_wd:N \l_tmpa_box }
1726            \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_right_delim_tl $ }
1727            \dim_set:Nn \l_@@_right_delim_dim { \box_wd:N \l_tmpa_box }
1728          }
1729          {
1730            \dim_gset:Nn \l_@@_left_delim_dim
1731              { 2 \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
1732            \dim_gset_eq:NN \l_@@_right_delim_dim \l_@@_left_delim_dim
1733          }
```

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).

```
1734        \hbox_set:Nw \l_@@_the_array_box
1735        \bool_if:NT \c_@@_testphase_table_bool
1736          { \UseTaggingSocket { tbl / hmode / begin } }
1737        \skip_horizontal:N \l_@@_left_margin_dim
1738        \skip_horizontal:N \l_@@_extra_left_margin_dim
1739        \c_math_toggle_token
1740        \bool_if:NTF \l_@@_light_syntax_bool
1741          { \use:c { @@-light-syntax } }
1742          { \use:c { @@-normal-syntax } }
1743      }
```

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

```
1744 \cs_new_protected_nopar:Npn \@@_CodeBefore_Body:w #1 \Body
1745   {
1746     \tl_set:Nn \l_tmpa_tl { #1 }
1747     \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
1748       { \@@_rescan_for_spanish:N \l_tmpa_tl }
1749     \tl_gput_left:NV \g_@@_pre_code_before_tl \l_tmpa_tl
1750     \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...

```
1751        \@@_pre_array:
1752      }
```

10   The \CodeBefore

The following command will be executed if the \CodeBefore has to be actually executed (that commmand will be used only once and is present alone only for legibility).

```
1753 \cs_new_protected:Npn \@@_pre_code_before:
1754   {
```

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).

```
1755     \int_set:Nn \c@iRow { \seq_item:Nn \g_@@_size_seq 2 }
1756     \int_set:Nn \c@jCol { \seq_item:Nn \g_@@_size_seq 5 }
1757     \int_set_eq:NN \g_@@_row_total_int { \seq_item:Nn \g_@@_size_seq 3 }
1758     \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.

```
1759     \pgfsys@markposition { \@@_env: - position }
1760     \pgfsys@getposition { \@@_env: - position } \@@_picture_position:
1761     \pgfpicture
1762     \pgf@relevantforpicturesizefalse
```

First, the recreation of the row nodes.

```
1763     \int_step_inline:nnn \l_@@_first_row_int { \g_@@_row_total_int + 1 }
1764       {
1765         \pgfsys@getposition { \@@_env: - row - ##1 } \@@_node_position:
1766         \pgfcoordinate { \@@_env: - row - ##1 }
1767           { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1768       }
```

Now, the recreation of the col nodes.

```
1769     \int_step_inline:nnn \l_@@_first_col_int { \g_@@_col_total_int + 1 }
1770       {
1771         \pgfsys@getposition { \@@_env: - col - ##1 } \@@_node_position:
1772         \pgfcoordinate { \@@_env: - col - ##1 }
1773           { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1774       }
```

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

```
1775     \@@_create_diag_nodes:
```

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

```
1776     \bool_if:NT \g_@@_recreate_cell_nodes_bool \@@_recreate_cell_nodes:
1777     \endpgfpicture
```

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

```
1778     \@@_create_blocks_nodes:
1779     \IfPackageLoadedTF { tikz }
1780       {
1781         \tikzset
1782           {
1783             every~picture / .style =
1784               { overlay , name~prefix = \@@_env: - }
1785           }
1786       }
1787       { }
1788     \cs_set_eq:NN \cellcolor \@@_cellcolor
1789     \cs_set_eq:NN \rectanglecolor \@@_rectanglecolor
1790     \cs_set_eq:NN \roundedrectanglecolor \@@_roundedrectanglecolor
1791     \cs_set_eq:NN \rowcolor \@@_rowcolor
```

```
1792    \cs_set_eq:NN \rowcolors \@@_rowcolors
1793    \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors
1794    \cs_set_eq:NN \arraycolor \@@_arraycolor
1795    \cs_set_eq:NN \columncolor \@@_columncolor
1796    \cs_set_eq:NN \chessboardcolors \@@_chessboardcolors
1797    \cs_set_eq:NN \SubMatrix \@@_SubMatrix_in_code_before
1798    \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
1799    \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
1800  }
```

```
1801 \cs_new_protected:Npn \@@_exec_code_before:
1802  {
1803    \seq_gclear_new:N \g_@@_colors_seq
```

The sequence `\g_@@_colors_seq` will always contain as first element the special color `nocolor`: when that color is used, no color will be applied in the corresponding cells by the other coloring commands of nicematrix.

```
1804    \@@_add_to_colors_seq:nn { { nocolor } } { }
1805    \bool_gset_false:N \g_@@_recreate_cell_nodes_bool
1806    \group_begin:
```

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

```
1807    \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 ASCCI 60) and > are activated and Tikz is not able to solve the problem (even with the Tikz library babel).

```
1808    \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
1809      { \@@_rescan_for_spanish:N \l_@@_code_before_tl }
```

Here is the `\CodeBefore`. The construction is a bit complicated because `\g_@@_pre_code_before_tl` may begin with keys between square brackets. Moreover, after the analyze of those keys, we sometimes have to decide to do *not* execute the rest of `\g_@@_pre_code_before_tl` (when it is asked for the creation of cell nodes in the `\CodeBefore`). That's why we use a `\q_stop`: it will be used to discard the rest of `\g_@@_pre_code_before_tl`.

```
1810    \exp_last_unbraced:NV \@@_CodeBefore_keys:
1811      \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.

```
1812    \@@_actually_color:
1813    \l_@@_code_before_tl
1814    \q_stop
1815  \bool_if:NT \l_@@_tabular_bool \c_math_toggle_token
1816  \group_end:
1817  \bool_if:NT \g_@@_recreate_cell_nodes_bool
1818    { \tl_put_left:Nn \@@_node_for_cell: \@@_patch_node_for_cell: }
1819  }
```

```
1820 \keys_define:nn { nicematrix / CodeBefore }
1821  {
1822    create-cell-nodes .bool_gset:N = \g_@@_recreate_cell_nodes_bool ,
1823    create-cell-nodes .default:n = true ,
1824    sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
1825    sub-matrix .value_required:n = true ,
1826    delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1827    delimiters / color .value_required:n = true ,
1828    unknown .code:n = \@@_error:n { Unknown~key~for~CodeBefore }
1829  }
```

```
1830 \NewDocumentCommand \@@_CodeBefore_keys: { O { } }
1831   {
1832     \keys_set:nn { nicematrix / CodeBefore } { #1 }
1833     \@@_CodeBefore:w
1834   }
```

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.

```
1835 \cs_new_protected:Npn \@@_CodeBefore:w #1 \q_stop
1836   {
1837     \bool_if:NT \g_@@_aux_found_bool
1838       {
1839         \@@_pre_code_before:
1840         #1
1841       }
1842   }
```

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.

```
1843 \cs_new_protected:Npn \@@_recreate_cell_nodes:
1844   {
1845     \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
1846       {
1847         \pgfsys@getposition { \@@_env: - ##1 - base } \@@_node_position:
1848         \pgfcoordinate { \@@_env: - row - ##1 - base }
1849           { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1850         \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
1851           {
1852             \cs_if_exist:cT
1853               { pgf @ sys @ pdf @ mark @ pos @ \@@_env: - ##1 - ####1 - NW }
1854               {
1855                 \pgfsys@getposition
1856                   { \@@_env: - ##1 - ####1 - NW }
1857                   \@@_node_position:
1858                 \pgfsys@getposition
1859                   { \@@_env: - ##1 - ####1 - SE }
1860                   \@@_node_position_i:
1861                 \@@_pgf_rect_node:nnn
1862                   { \@@_env: - ##1 - ####1 }
1863                   { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1864                   { \pgfpointdiff \@@_picture_position: \@@_node_position_i: }
1865               }
1866           }
1867       }
1868     \int_step_inline:nn \c@iRow
1869       {
1870         \pgfnodealias
1871           { \@@_env: - ##1 - last }
1872           { \@@_env: - ##1 - \int_use:N \c@jCol }
1873       }
1874     \int_step_inline:nn \c@jCol
1875       {
1876         \pgfnodealias
1877           { \@@_env: - last - ##1 }
1878           { \@@_env: - \int_use:N \c@iRow - ##1 }
1879       }
1880     \@@_create_extra_nodes:
1881   }
```

```
1882  \cs_new_protected:Npn \@@_create_blocks_nodes:
1883    {
1884      \pgfpicture
1885      \pgf@relevantforpicturesizefalse
1886      \pgfrememberpicturepositiononpagetrue
1887      \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
1888        { \@@_create_one_block_node:nnnnn ##1 }
1889      \endpgfpicture
1890    }
```

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]

```
1891  \cs_new_protected:Npn \@@_create_one_block_node:nnnnn #1 #2 #3 #4 #5
1892    {
1893      \tl_if_empty:nF { #5 }
1894        {
1895          \@@_qpoint:n { col - #2 }
1896          \dim_set_eq:NN \l_tmpa_dim \pgf@x
1897          \@@_qpoint:n { #1 }
1898          \dim_set_eq:NN \l_tmpb_dim \pgf@y
1899          \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
1900          \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
1901          \@@_qpoint:n { \int_eval:n { #3 + 1 } }
1902          \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
1903          \@@_pgf_rect_node:nnnnn
1904            { \@@_env: - #5 }
1905            { \dim_use:N \l_tmpa_dim }
1906            { \dim_use:N \l_tmpb_dim }
1907            { \dim_use:N \l_@@_tmpc_dim }
1908            { \dim_use:N \l_@@_tmpd_dim }
1909        }
1910    }


1911  \cs_new_protected:Npn \@@_patch_for_revtex:
1912    {
1913      \cs_set_eq:NN \@addamp \@addamp@LaTeX
1914      \cs_set_eq:NN \insert@column \insert@column@array
1915      \cs_set_eq:NN \@classx \@classx@array
1916      \cs_set_eq:NN \@xarraycr \@xarraycr@array
1917      \cs_set_eq:NN \@arraycr \@arraycr@array
1918      \cs_set_eq:NN \@xargarraycr \@xargarraycr@array
1919      \cs_set_eq:NN \array \array@array
1920      \cs_set_eq:NN \@array \@array@array
1921      \cs_set_eq:NN \@tabular \@tabular@array
1922      \cs_set_eq:NN \@mkpream \@mkpream@array
1923      \cs_set_eq:NN \endarray \endarray@array
1924      \cs_set:Npn \@tabarray { \@ifnextchar [ { \@array } { \@array [ c ] } }
1925      \cs_set:Npn \endtabular { \endarray $\egroup} % $
1926    }
```

11   The environment {NiceArrayWithDelims}

```
1927  \NewDocumentEnvironment { NiceArrayWithDelims }
1928    { m m O { } m ! O { } t \CodeBefore }
1929    {
```

---

[6]Moreover, 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).

```
1930    \bool_if:NT \c_@@_revtex_bool \@@_patch_for_revtex:
1931    \@@_provide_pgfsyspdfmark:
1932    \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.

```
1933    \bgroup

1934    \tl_gset:Nn \g_@@_left_delim_tl { #1 }
1935    \tl_gset:Nn \g_@@_right_delim_tl { #2 }
1936    \tl_gset:Nn \g_@@_user_preamble_tl { #4 }
1937    \tl_if_empty:NT \g_@@_user_preamble_tl { \@@_fatal:n { empty~preamble } }

1938    \int_gzero:N \g_@@_block_box_int
1939    \dim_zero:N \g_@@_width_last_col_dim
1940    \dim_zero:N \g_@@_width_first_col_dim
1941    \bool_gset_false:N \g_@@_row_of_col_done_bool
1942    \str_if_empty:NT \g_@@_name_env_str
1943      { \str_gset:Nn \g_@@_name_env_str { NiceArrayWithDelims } }
1944    \bool_if:NTF \l_@@_tabular_bool
1945      \mode_leave_vertical:
1946      \@@_test_if_math_mode:
1947    \bool_if:NT \l_@@_in_env_bool { \@@_fatal:n { Yet~in~env } }
1948    \bool_set_true:N \l_@@_in_env_bool
```

The command \CT@arc@ contains the instruction of color for the rules of the array[7]. This command is used by \CT@arc@ but we use it also for compatibility with colortbl. But we want also to be able to use color for the rules of the array when colortbl is *not* loaded. That's why we do the following instruction which is in the patch of the beginning of arrays done by colortbl. Of course, we restore the value of \CT@arc@ at the end of our environment.

```
1949    \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.

```
1950    \cs_if_exist:NT \tikz@library@external@loaded
1951      {
1952        \tikzexternaldisable
1953        \cs_if_exist:NT \ifstandalone
1954          { \tikzset { external / optimize = false } }
1955      }
```

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

```
1956    \int_gincr:N \g_@@_env_int
1957    \bool_if:NF \l_@@_block_auto_columns_width_bool
1958      { \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).

```
1959    \seq_gclear:N \g_@@_blocks_seq
1960    \seq_gclear:N \g_@@_pos_of_blocks_seq
```

In fact, the sequence \g_@@_pos_of_blocks_seq will also contain the positions of the cells with a \diagbox and the \multicolumn.

```
1961    \seq_gclear:N \g_@@_pos_of_stroken_blocks_seq
1962    \seq_gclear:N \g_@@_pos_of_xdots_seq
1963    \tl_gclear_new:N \g_@@_code_before_tl
1964    \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.

---

[7]e.g. \color[rgb]{0.5,0.5,0}

```
1965        \tl_if_exist:cTF { c_@@ _ \int_use:N \g_@@_env_int _ tl }
1966          {
1967            \bool_gset_true:N \g_@@_aux_found_bool
1968            \use:c { c_@@ _ \int_use:N \g_@@_env_int _ tl }
1969          }
1970          { \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.

```
1971        \tl_gclear:N \g_@@_aux_tl
1972        \tl_if_empty:NF \g_@@_code_before_tl
1973          {
1974            \bool_set_true:N \l_@@_code_before_bool
1975            \tl_put_right:NV \l_@@_code_before_tl \g_@@_code_before_tl
1976          }
1977        \tl_if_empty:NF \g_@@_pre_code_before_tl
1978          { \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.

```
1979        \bool_if:NTF \g_@@_delims_bool
1980          { \keys_set:nn { nicematrix / pNiceArray } }
1981          { \keys_set:nn { nicematrix / NiceArray } }
1982        { #3 , #5 }


1983        \@@_set_CT@arc@:o \l_@@_rules_color_tl

```

The argument #6 is the last argument of {NiceArrayWithDelims}. With that argument of type "t \CodeBefore", we test whether there is the keyword \CodeBefore at the beginning of the body of the environment. If that keyword is present, we have now to extract all the content between that keyword \CodeBefore and the (other) keyword \Body. It's the job that will do the command \@@_CodeBefore_Body:w. After that job, the command \@@_CodeBefore_Body:w will go on with \@@_pre_array:.

```
1984        \bool_if:nTF { #6 } \@@_CodeBefore_Body:w \@@_pre_array:
1985      }
```

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

```
1986      {
1987      \bool_if:NTF \l_@@_light_syntax_bool
1988        { \use:c { end @@-light-syntax } }
1989        { \use:c { end @@-normal-syntax } }
1990      \c_math_toggle_token
1991      \skip_horizontal:N \l_@@_right_margin_dim
1992      \skip_horizontal:N \l_@@_extra_right_margin_dim
1993
1994      % awful workaround
1995      \int_compare:nNnT \g_@@_col_total_int = \c_one_int
1996        {
1997          \dim_compare:nNnT \l_@@_columns_width_dim > \c_zero_dim
1998            {
1999              \skip_horizontal:N - \l_@@_columns_width_dim
2000              \bool_if:NTF \l_@@_tabular_bool
2001                { \skip_horizontal:n { - 2 \tabcolsep } }
2002                { \skip_horizontal:n { - 2 \arraycolsep } }
2003            }
2004        }
2005      \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.

```
2006        \bool_if:NT \l_@@_width_used_bool
2007          {
```

```
2008        \int_if_zero:nT \g_@@_total_X_weight_int
2009          { \@@_error_or_warning:n { width~without~X~columns } }
2010        }
```

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.

```
2011        \int_compare:nNnT \g_@@_total_X_weight_int > \c_zero_int
2012          {
2013            \tl_gput_right:Ne \g_@@_aux_tl
2014              {
2015                \bool_set_true:N \l_@@_X_columns_aux_bool
2016                \dim_set:Nn \l_@@_X_columns_dim
2017                  {
2018                    \dim_compare:nNnTF
2019                      {
2020                        \dim_abs:n
2021                          { \l_@@_width_dim - \box_wd:N \l_@@_the_array_box }
2022                      }
2023                      <
2024                      { 0.001 pt }
2025                      { \dim_use:N \l_@@_X_columns_dim }
2026                      {
2027                        \dim_eval:n
2028                          {
2029                            ( \l_@@_width_dim - \box_wd:N \l_@@_the_array_box )
2030                            / \int_use:N \g_@@_total_X_weight_int
2031                            + \l_@@_X_columns_dim
2032                          }
2033                      }
2034                  }
2035              }
2036          }
```

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

```
2037        \int_compare:nNnT \l_@@_last_row_int > { -2 }
2038          {
2039            \bool_if:NF \l_@@_last_row_without_value_bool
2040              {
2041                \int_compare:nNnF \l_@@_last_row_int = \c@iRow
2042                  {
2043                    \@@_error:n { Wrong~last~row }
2044                    \int_gset_eq:NN \l_@@_last_row_int \c@iRow
2045                  }
2046              }
2047          }
```

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]

```
2048        \int_gset_eq:NN \c@jCol \g_@@_col_total_int
2049        \bool_if:NTF \g_@@_last_col_found_bool
2050          { \int_gdecr:N \c@jCol }
2051          {
2052            \int_compare:nNnT \l_@@_last_col_int > { -1 }
2053              { \@@_error:n { last~col~not~used } }
2054          }
```

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

```
2055        \int_gset_eq:NN \g_@@_row_total_int \c@iRow
```

---

[8]We remind that the potential "first column" (exterior) has the number 0.

```
2056        \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.).

```
2057        \int_if_zero:nT \l_@@_first_col_int
2058          { \skip_horizontal:N \g_@@_width_first_col_dim }
```

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

```
2059        \bool_if:nTF { ! \g_@@_delims_bool }
2060          {
2061            \tl_if_eq:NNTF \l_@@_baseline_tl \c_@@_c_tl
2062              \@@_use_arraybox_with_notes_c:
2063              {
2064                \tl_if_eq:NNTF \l_@@_baseline_tl \c_@@_b_tl
2065                  \@@_use_arraybox_with_notes_b:
2066                  \@@_use_arraybox_with_notes:
2067              }
2068          }
```

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).

```
2069        {
2070          \int_if_zero:nTF \l_@@_first_row_int
2071            {
2072              \dim_set_eq:NN \l_tmpa_dim \g_@@_dp_row_zero_dim
2073              \dim_add:Nn \l_tmpa_dim \g_@@_ht_row_zero_dim
2074            }
2075            { \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]

```
2076          \int_compare:nNnTF \l_@@_last_row_int > { -2 }
2077            {
2078              \dim_set_eq:NN \l_tmpb_dim \g_@@_ht_last_row_dim
2079              \dim_add:Nn \l_tmpb_dim \g_@@_dp_last_row_dim
2080            }
2081            { \dim_zero:N \l_tmpb_dim }
2082          \hbox_set:Nn \l_tmpa_box
2083            {
2084              \c_math_toggle_token
2085              \@@_color:o \l_@@_delimiters_color_tl
2086              \exp_after:wN \left \g_@@_left_delim_tl
2087              \vcenter
2088                {
```

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.

```
2089                  \skip_vertical:n { -\l_tmpa_dim - \arrayrulewidth }
2090                  \hbox
2091                    {
2092                      \bool_if:NTF \l_@@_tabular_bool
2093                        { \skip_horizontal:N -\tabcolsep }
2094                        { \skip_horizontal:N -\arraycolsep }
2095                      \@@_use_arraybox_with_notes_c:
2096                      \bool_if:NTF \l_@@_tabular_bool
2097                        { \skip_horizontal:N -\tabcolsep }
2098                        { \skip_horizontal:N -\arraycolsep }
2099                    }
```

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

```
2100                  \skip_vertical:N -\l_tmpb_dim
```

---

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

```
2101                \skip_vertical:N \arrayrulewidth
2102            }
2103          \exp_after:wN \right \g_@@_right_delim_tl
2104          \c_math_toggle_token
2105        }
```

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.

```
2106        \bool_if:NTF \l_@@_delimiters_max_width_bool
2107          {
2108            \@@_put_box_in_flow_bis:nn
2109              \g_@@_left_delim_tl
2110              \g_@@_right_delim_tl
2111          }
2112          \@@_put_box_in_flow:
2113      }
```

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.).

```
2114      \bool_if:NT \g_@@_last_col_found_bool
2115        { \skip_horizontal:N \g_@@_width_last_col_dim }
2116      \bool_if:NT \l_@@_preamble_bool
2117        {
2118          \int_compare:nNnT \c@jCol < \g_@@_static_num_of_col_int
2119            { \@@_warning_gredirect_none:n  { columns~not~used } }
2120        }
2121      \@@_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.

```
2122      \egroup
```

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

```
2123      \iow_now:Nn \@mainaux { \ExplSyntaxOn }
2124      \iow_now:Nn \@mainaux { \char_set_catcode_space:n { 32 }  }
2125      \iow_now:Ne \@mainaux
2126        {
2127          \tl_gset:cn { c_@@_ \int_use:N \g_@@_env_int _ tl }
2128            { \exp_not:o \g_@@_aux_tl }
2129        }
2130      \iow_now:Nn \@mainaux { \ExplSyntaxOff }

2131      \bool_if:NT \g_@@_footnote_bool \endsavenotes
2132    }
```

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

12   We construct the preamble of the array

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

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

```
2133 \cs_new_protected:Npn \@@_transform_preamble:
2134   {
2135     \@@_transform_preamble_i:
2136     \@@_transform_preamble_ii:
2137   }
```

```
2138  \cs_new_protected:Npn \@@_transform_preamble_i:
2139    {
2140      \int_gzero:N \c@jCol
```

The sequence \g_@@_cols_vlsim_seq will contain the numbers of the columns where you will to have to draw vertical lines in the potential sub-matrices (hence the name vlism).

```
2141      \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.

```
2142      \bool_gset_false:N \g_tmpb_bool
```

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

```
2143      \tl_gclear_new:N \g_@@_pre_cell_tl
```

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

```
2144      \int_zero:N \l_tmpa_int
2145      \tl_gclear:N \g_@@_array_preamble_tl
2146      \tl_if_eq:NNTF \l_@@_vlines_clist \c_@@_all_tl
2147        {
2148          \tl_gset:Nn \g_@@_array_preamble_tl
2149            { ! { \skip_horizontal:N \arrayrulewidth } }
2150        }
2151        {
2152          \clist_if_in:NnT \l_@@_vlines_clist 1
2153            {
2154              \tl_gset:Nn \g_@@_array_preamble_tl
2155                { ! { \skip_horizontal:N \arrayrulewidth } }
2156            }
2157        }
```

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

```
2158      \exp_last_unbraced:NV \@@_rec_preamble:n \g_@@_user_preamble_tl \stop
2159      \int_gset_eq:NN \g_@@_static_num_of_col_int \c@jCol

2160      \@@_replace_columncolor:
2161    }


2162  \hook_gput_code:nnn { begindocument } { . }
2163    {
2164      \IfPackageLoadedTF { colortbl }
2165        {
```

When colortbl is used, we have to catch the tokens \columncolor in the preamble because, otherwise, colortbl will catch them and the colored panels won't be drawn by nicematrix but by colortbl (with an output which is not perfect).

```
2166          \regex_const:Nn \c_@@_columncolor_regex { \c { columncolor } }
2167          \cs_new_protected:Npn \@@_replace_columncolor:
2168            {
2169              \regex_replace_all:NnN
2170                \c_@@_columncolor_regex
2171                { \c { @@_columncolor_preamble } }
2172                \g_@@_array_preamble_tl
2173            }
2174        }
2175        {
2176          \cs_new_protected:Npn \@@_replace_columncolor:
2177            { \cs_set_eq:NN \columncolor \@@_columncolor_preamble }
2178        }
2179    }


2180  \cs_new_protected:Npn \@@_transform_preamble_ii:
2181    {
```

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

```
2182        \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2183          {
2184            \tl_if_eq:NNF \g_@@_right_delim_tl \c_@@_dot_tl
2185              { \bool_gset_true:N \g_@@_delims_bool }
2186          }
2187          { \bool_gset_true:N \g_@@_delims_bool }
```

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

```
2188        \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).

```
2189        \int_if_zero:nTF \l_@@_first_col_int
2190          { \tl_gput_left:No \g_@@_array_preamble_tl \c_@@_preamble_first_col_tl }
2191          {
2192            \bool_if:NF \g_@@_delims_bool
2193              {
2194                \bool_if:NF \l_@@_tabular_bool
2195                  {
2196                    \tl_if_empty:NT \l_@@_vlines_clist
2197                      {
2198                        \bool_if:NF \l_@@_exterior_arraycolsep_bool
2199                          { \tl_gput_left:Nn \g_@@_array_preamble_tl { @ { } } } }
2200                  }
2201              }
2202          }
2203          }
2204        \int_compare:nNnTF \l_@@_last_col_int > { -1 }
2205          { \tl_gput_right:No \g_@@_array_preamble_tl \c_@@_preamble_last_col_tl }
2206          {
2207            \bool_if:NF \g_@@_delims_bool
2208              {
2209                \bool_if:NF \l_@@_tabular_bool
2210                  {
2211                    \tl_if_empty:NT \l_@@_vlines_clist
2212                      {
2213                        \bool_if:NF \l_@@_exterior_arraycolsep_bool
2214                          { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { } } } }
2215                  }
2216              }
2217          }
2218          }
```

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).

```
2219        \dim_compare:nNnT \l_@@_tabular_width_dim = \c_zero_dim
2220          {
2221            \tl_gput_right:Nn \g_@@_array_preamble_tl
2222              { > { \@@_error_too_much_cols: } l }
2223          }
2224    }
```

The preamble provided by the final user will be read by a finite automata. The following function \@@_rec_preamble:n will read that preamble (usually letter by letter) in a recursive way (hence the name of that function). in the preamble.

```
2225 \cs_new_protected:Npn \@@_rec_preamble:n #1
2226    {
```

For the majority of the letters, we will trigger the corresponding action by calling directly a function in the main hashtable of TeX (thanks to the mechanism \csname...\endcsname. Be careful: all these functions take in as first argument the letter (or token) itself.[10]

```
2227        \cs_if_exist:cTF { @@ _ \token_to_str:N #1 }
2228          { \use:c { @@ _ \token_to_str:N #1 } { #1 } }
2229          {
```

Now, the columns defined by \newcolumntype of array.

```
2230            \cs_if_exist:cTF { NC @ find @ #1 }
2231              {
2232                \tl_set_eq:Nc \l_tmpb_tl { NC @ rewrite @ #1 }
2233                \exp_last_unbraced:NV \@@_rec_preamble:n \l_tmpb_tl
2234              }
2235              {
2236                \tl_if_eq:nnT { #1 } { S }
2237                  { \@@_fatal:n { unknown~column~type~S } }
2238                  { \@@_fatal:nn { unknown~column~type } { #1 } }
2239              }
2240          }
2241      }
```

For c, l and r

```
2242  \cs_new:Npn \@@_c #1
2243    {
2244      \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2245      \tl_gclear:N \g_@@_pre_cell_tl
2246      \tl_gput_right:Nn \g_@@_array_preamble_tl
2247        { > \@@_cell_begin:w c < \@@_cell_end: }
```

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

```
2248        \int_gincr:N \c@jCol
2249        \@@_rec_preamble_after_col:n
2250    }
2251  \cs_new:Npn \@@_l #1
2252    {
2253      \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2254      \tl_gclear:N \g_@@_pre_cell_tl
2255      \tl_gput_right:Nn \g_@@_array_preamble_tl
2256        {
2257          > { \@@_cell_begin:w \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_l_tl }
2258          l
2259          < \@@_cell_end:
2260        }
2261      \int_gincr:N \c@jCol
2262      \@@_rec_preamble_after_col:n
2263    }
2264  \cs_new:Npn \@@_r #1
2265    {
2266      \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2267      \tl_gclear:N \g_@@_pre_cell_tl
2268      \tl_gput_right:Nn \g_@@_array_preamble_tl
2269        {
2270          > { \@@_cell_begin:w \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_r_tl }
2271          r
2272          < \@@_cell_end:
2273        }
2274      \int_gincr:N \c@jCol
2275      \@@_rec_preamble_after_col:n
2276    }
```

---

[10]We do that because it's an easy way to insert the letter at some places in the code that we will add to \g_@@_array_preamble_tl.

For ! and @

```
2277 \cs_new:cpn { @@ _ \token_to_str:N ! } #1 #2
2278   {
2279     \tl_gput_right:Nn \g_@@_array_preamble_tl { #1 { #2 } }
2280     \@@_rec_preamble:n
2281   }
2282 \cs_set_eq:cc { @@ _ \token_to_str:N @ } { @@ _ \token_to_str:N ! }
```

For |

```
2283 \cs_new:cpn { @@ _ | } #1
2284   {
```

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

```
2285     \int_incr:N \l_tmpa_int
2286     \@@_make_preamble_i_i:n
2287   }
2288 \cs_new_protected:Npn \@@_make_preamble_i_i:n #1
2289   {
2290     \str_if_eq:nnTF { #1 } |
2291       { \use:c { @@ _ | } | }
2292       { \@@_make_preamble_i_ii:nn { } #1 }
2293   }
2294 \cs_new_protected:Npn \@@_make_preamble_i_ii:nn #1 #2
2295   {
2296     \str_if_eq:nnTF { #2 } [
2297       { \@@_make_preamble_i_ii:nw { #1 } [ }
2298       { \@@_make_preamble_i_iii:nn { #2 } { #1 } }
2299   }
2300 \cs_new_protected:Npn \@@_make_preamble_i_ii:nw #1 [ #2 ]
2301   { \@@_make_preamble_i_ii:nn { #1 , #2 } }
2302 \cs_new_protected:Npn \@@_make_preamble_i_iii:nn #1 #2
2303   {
2304     \@@_compute_rule_width:n { multiplicity = \l_tmpa_int , #2 }
2305     \tl_gput_right:Ne \g_@@_array_preamble_tl
2306       {
```

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

```
2307         \exp_not:N ! { \skip_horizontal:n { \dim_eval:n { \l_@@_rule_width_dim } } } }
2308       }
2309     \tl_gput_right:Ne \g_@@_pre_code_after_tl
2310       {
2311         \@@_vline:n
2312           {
2313             position = \int_eval:n { \c@jCol + 1 } ,
2314             multiplicity = \int_use:N \l_tmpa_int ,
2315             total-width = \dim_use:N \l_@@_rule_width_dim ,
2316             #2
2317           }
```

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.

```
2318       }
2319     \int_zero:N \l_tmpa_int
2320     \str_if_eq:nnT { #1 } { \stop } { \bool_gset_true:N \g_tmpb_bool }
2321     \@@_rec_preamble:n #1
2322   }


2323 \cs_new:cpn { @@ _ > } #1 #2
2324   {
2325     \tl_gput_right:Nn \g_@@_pre_cell_tl { > { #2 } }
2326     \@@_rec_preamble:n
2327   }
```

```
2328  \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.

```
2329  \keys_define:nn { nicematrix / p-column }
2330    {
2331      r .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_r_str ,
2332      r .value_forbidden:n = true ,
2333      c .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_c_str ,
2334      c .value_forbidden:n = true ,
2335      l .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_l_str ,
2336      l .value_forbidden:n = true ,
2337      R .code:n =
2338        \IfPackageLoadedTF { ragged2e }
2339          { \str_set_eq:NN \l_@@_hpos_col_str \c_@@_R_str }
2340          {
2341            \@@_error_or_warning:n { ragged2e~not~loaded }
2342            \str_set_eq:NN \l_@@_hpos_col_str \c_@@_r_str
2343          } ,
2344      R .value_forbidden:n = true ,
2345      L .code:n =
2346        \IfPackageLoadedTF { ragged2e }
2347          { \str_set_eq:NN \l_@@_hpos_col_str \c_@@_L_stsr }
2348          {
2349            \@@_error_or_warning:n { ragged2e~not~loaded }
2350            \str_set_eq:NN \l_@@_hpos_col_str \c_@@_l_str
2351          } ,
2352      L .value_forbidden:n = true ,
2353      C .code:n =
2354        \IfPackageLoadedTF { ragged2e }
2355          { \str_set_eq:NN \l_@@_hpos_col_str \c_@@_C_str }
2356          {
2357            \@@_error_or_warning:n { ragged2e~not~loaded }
2358            \str_set_eq:NN \l_@@_hpos_col_str \c_@@_c_str
2359          } ,
2360      C .value_forbidden:n = true ,
2361      S .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_si_str ,
2362      S .value_forbidden:n = true ,
2363      p .code:n = \str_set:Nn \l_@@_vpos_col_str { p } ,
2364      p .value_forbidden:n = true ,
2365      t .meta:n = p ,
2366      m .code:n = \str_set:Nn \l_@@_vpos_col_str { m } ,
2367      m .value_forbidden:n = true ,
2368      b .code:n = \str_set:Nn \l_@@_vpos_col_str { b } ,
2369      b .value_forbidden:n = true ,
2370    }
```

For p but also b and m.

```
2371  \cs_new:Npn \@@_p #1
2372    {
2373      \str_set:Nn \l_@@_vpos_col_str { #1 }
```

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

```
2374      \@@_make_preamble_ii_i:n
2375    }
2376  \cs_set_eq:NN \@@_b \@@_p
2377  \cs_set_eq:NN \@@_m \@@_p

2378  \cs_new_protected:Npn \@@_make_preamble_ii_i:n #1
2379    {
2380      \str_if_eq:nnTF { #1 } { [ }
2381        { \@@_make_preamble_ii_ii:w [ }
2382        { \@@_make_preamble_ii_ii:w [ ] { #1 } } }
2383    }
```

```
2384  \cs_new_protected:Npn \@@_make_preamble_ii_ii:w [ #1 ]
2385    { \@@_make_preamble_ii_iii:nn { #1 } }
```

#1 is the optional argument of the specifier (a list of *key-value* pairs).
#2 is the mandatory argument of the specifier: the width of the column.

```
2386  \cs_new_protected:Npn \@@_make_preamble_ii_iii:nn #1 #2
2387    {
```

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).

```
2388      \str_set_eq:NN \l_@@_hpos_col_str \c_@@_j_str
2389      \@@_keys_p_column:n { #1 }
2390      \@@_make_preamble_ii_iv:nnn { #2 } { minipage } { }
2391    }
2392  \cs_new_protected:Npn \@@_keys_p_column:n #1
2393    { \keys_set_known:nnN { nicematrix / p-column } { #1 } \l_tmpa_tl }
```

The first argument is the width of the column. The second is the type of environment: minipage or
varwidth. The third is some code added at the beginning of the cell.

```
2394  \cs_new_protected:Npn \@@_make_preamble_ii_iv:nnn #1 #2 #3
2395    {
2396      \use:e
2397        {
2398          \@@_make_preamble_ii_v:nnnnnnnn
2399            { \str_if_eq:onTF \l_@@_vpos_col_str { p } { t } { b } }
2400            { \dim_eval:n { #1 } }
2401            {
```

The parameter \l_@@_hpos_col_str (as \l_@@_vpos_col_str) exists only during the construction
of the preamble. During the composition of the array itself, you will have, in each cell, the parameter
\l_@@_hpos_cell_tl which will provide the horizontal alignment of the column to which belongs
the cell.

```
2402              \str_if_eq:NNTF \l_@@_hpos_col_str \c_@@_j_str
2403                { \tl_clear:N \exp_not:N \l_@@_hpos_cell_tl }
2404                {
2405                  \cs_set_nopar:Npn \exp_not:N \l_@@_hpos_cell_tl
2406                    { \str_lowercase:o \l_@@_hpos_col_str }
2407                }
2408              \str_case:on \l_@@_hpos_col_str
2409                {
2410                  c { \exp_not:N \centering }
2411                  l { \exp_not:N \raggedright }
2412                  r { \exp_not:N \raggedleft }
2413                  C { \exp_not:N \Centering }
2414                  L { \exp_not:N \RaggedRight }
2415                  R { \exp_not:N \RaggedLeft }
2416                }
2417              #3
2418            }
2419            { \str_if_eq:onT \l_@@_vpos_col_str { m } \@@_center_cell_box: }
2420            { \str_if_eq:onT \l_@@_hpos_col_str { si } \siunitx_cell_begin:w }
2421            { \str_if_eq:onT \l_@@_hpos_col_str { si } \siunitx_cell_end: }
2422            { #2 }
2423            {
2424              \str_case:onF \l_@@_hpos_col_str
2425                {
2426                  { j } { c }
2427                  { si } { c }
2428                }
```

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

```
2429                { \str_lowercase:o \l_@@_hpos_col_str }
2430            }
2431          }
```

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

```
2432      \int_gincr:N \c@jCol
2433      \@@_rec_preamble_after_col:n
2434    }
```

#1 is the optional argument of {minipage} (or {varwidth}): t or b. Indeed, for the columns of type m, we use the value b here because there is a special post-action in order to center vertically the box (see #4).

#2 is the width of the {minipage} (or {varwidth}), that is to say also the width of the column.

#3 is the coding for the horizontal position of the content of the cell (\centering, \raggedright, \raggedleft or nothing). It's also possible to put in that #3 some code to fix the value of \l_@@_hpos_cell_tl which will be available in each cell of the column.

#4 is an extra-code which contains \@@_center_cell_box: (when the column is a m column) or nothing (in the other cases).

#5 is a code put just before the c (or r or l: see #8).

#6 is a code put just after the c (or r or l: see #8).

#7 is the type of environment: minipage or varwidth.

#8 is the letter c or r or l which is the basic specificier of column which is used *in fine*.

```
2435 \cs_new_protected:Npn \@@_make_preamble_ii_v:nnnnnnnn #1 #2 #3 #4 #5 #6 #7 #8
2436   {
2437     \tl_if_eq:NNTF \l_@@_hpos_col_str \c_@@_si_str
2438       { \tl_gput_right:Nn \g_@@_array_preamble_tl { > { \@@_test_if_empty_for_S: } } }
2439       { \tl_gput_right:Nn \g_@@_array_preamble_tl { > { \@@_test_if_empty: } } }
2440     \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2441     \tl_gclear:N \g_@@_pre_cell_tl
2442     \tl_gput_right:Nn \g_@@_array_preamble_tl
2443       {
2444         > {
```

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

```
2445           \dim_set:Nn \l_@@_col_width_dim { #2 }
2446           \bool_if:NT \c_@@_testphase_table_bool
2447             { \tag_struct_begin:n { tag = Div } }
2448           \@@_cell_begin:w
```

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

```
2449           \use:c { #7 } [ #1 ] { #2 }
```

The following lines have been taken from array.sty.

```
2450           \everypar
2451             {
2452               \vrule height \box_ht:N \@arstrutbox width \c_zero_dim
2453               \everypar { }
2454             }
2455           \bool_if:NT \c_@@_testphase_table_bool \tagpdfparaOn
```

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

```
2456           #3
```

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

```
2457           \g_@@_row_style_tl
2458           \arraybackslash
2459           #5
2460         }
2461       #8
2462       < {
2463         #6
```

The following line has been taken from array.sty.

```
2464         \@finalstrut \@arstrutbox
2465         \use:c { end #7 }
```

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

```
2466              #4
2467              \@@_cell_end:
2468              \bool_if:NT \c_@@_testphase_table_bool \tag_struct_end:
2469            }
2470        }
2471    }
```

```
2472 \str_new:N \c_@@_ignorespaces_str
2473 \str_set:Ne \c_@@_ignorespaces_str { \ignorespaces }
2474 \str_remove_all:Nn \c_@@_ignorespaces_str { ~ }
2475 \cs_new_protected:Npn \@@_test_if_empty:
2476    { \peek_after:Nw \@@_test_if_empty_i: }
2477 \cs_new_protected:Npn \@@_test_if_empty_i:
2478    {
2479      \str_set:Ne \l_tmpa_str { \token_to_meaning:N \l_peek_token }
2480      \str_if_eq:NNT \l_tmpa_str \c_@@_ignorespaces_str
2481        { \@@_test_if_empty:w }
2482    }
2483 \cs_new_protected:Npn \@@_test_if_empty:w \ignorespaces
2484    { \peek_after:Nw \@@_test_if_empty_ii: }
```

```
2485 \cs_new_protected:Npn \@@_nullify_cell:
2486    {
2487      \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2488        {
2489          \box_set_wd:Nn \l_@@_cell_box \c_zero_dim
2490          \skip_horizontal:N \l_@@_col_width_dim
2491        }
2492    }
```

```
2493 \bool_if:NTF \c_@@_tagging_array_bool
2494    {
2495      \cs_new_protected:Npn \@@_test_if_empty_ii:
2496        { \peek_meaning:NT \textonly@unskip \@@_nullify_cell: }
2497    }
```

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

```
2498    {
2499      \cs_new_protected:Npn \@@_test_if_empty_ii:
2500        { \peek_meaning:NT \unskip \@@_nullify_cell: }
2501    }
2502 \cs_new_protected:Npn \@@_test_if_empty_for_S:
2503    {
2504      \peek_meaning:NT \__siunitx_table_skip:n
2505        {
2506          \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2507            { \box_set_wd:Nn \l_@@_cell_box \c_zero_dim }
2508        }
2509    }
```

The following command will be used in m-columns in order to center vertically the box. In fact, despite its name, the command does not always center the cell. Indeed, if there is only one row in the cell, it should not be centered vertically. It's not possible to know the number of rows of the cell. However, we consider (as in array) that if the height of the cell is no more that the height of \strutbox, there is only one row.

```
2510 \cs_new_protected:Npn \@@_center_cell_box:
2511    {
```

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

```
2512      \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2513        {
2514          \int_compare:nNnT
2515            { \box_ht:N \l_@@_cell_box }
2516            >
```

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).

```
2517            { \box_ht:N \strutbox }
2518            {
2519              \hbox_set:Nn \l_@@_cell_box
2520                {
2521                  \box_move_down:nn
2522                    {
2523                      ( \box_ht:N \l_@@_cell_box - \box_ht:N \@arstrutbox
2524                      + \baselineskip ) / 2
2525                    }
2526                    { \box_use:N \l_@@_cell_box }
2527                }
2528            }
2529        }
2530    }
```

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

```
2531 \cs_new:Npn \@@_V #1 #2
2532    {
2533      \str_if_eq:nnTF { #2 } { [ }
2534        { \@@_make_preamble_V_i:w [ }
2535        { \@@_make_preamble_V_i:w [ ] { #2 } }
2536    }
2537 \cs_new_protected:Npn \@@_make_preamble_V_i:w [ #1 ]
2538    { \@@_make_preamble_V_ii:nn { #1 } }
2539 \cs_new_protected:Npn \@@_make_preamble_V_ii:nn #1 #2
2540    {
2541      \str_set:Nn \l_@@_vpos_col_str { p }
2542      \str_set_eq:NN \l_@@_hpos_col_str \c_@@_j_str
2543      \@@_keys_p_column:n { #1 }
2544      \IfPackageLoadedTF { varwidth }
2545        { \@@_make_preamble_ii_iv:nnn { #2 } { varwidth } { } }
2546        {
2547          \@@_error_or_warning:n { varwidth~not~loaded }
2548          \@@_make_preamble_ii_iv:nnn { #2 } { minipage } { }
2549        }
2550    }
```

For `w` and `W`

```
2551 \cs_new:Npn \@@_w { \@@_make_preamble_w:nnnn { } }
2552 \cs_new:Npn \@@_W { \@@_make_preamble_w:nnnn { \@@_special_W: } }
```

#1 is a special argument: empty for `w` and equal to `\@@_special_W:` for `W`;
#2 is the type of column (`w` or `W`);
#3 is the type of horizontal alignment (`c`, `l`, `r` or `s`);
#4 is the width of the column.

```
2553 \cs_new_protected:Npn \@@_make_preamble_w:nnnn #1 #2 #3 #4
2554    {
2555      \str_if_eq:nnTF { #3 } { s }
2556        { \@@_make_preamble_w_i:nnnn { #1 } { #4 } }
2557        { \@@_make_preamble_w_ii:nnnn { #1 } { #2 } { #3 } { #4 } }
2558    }
```

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.

```
2559  \cs_new_protected:Npn \@@_make_preamble_w_i:nnnn #1 #2
2560    {
2561      \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2562      \tl_gclear:N \g_@@_pre_cell_tl
2563      \tl_gput_right:Nn \g_@@_array_preamble_tl
2564        {
2565          > {
2566              \dim_set:Nn \l_@@_col_width_dim { #2 }
2567              \@@_cell_begin:w
2568              \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_c_tl
2569            }
2570          c
2571          < {
2572              \@@_cell_end_for_w_s:
2573              #1
2574              \@@_adjust_size_box:
2575              \box_use_drop:N \l_@@_cell_box
2576            }
2577        }
2578      \int_gincr:N \c@jCol
2579      \@@_rec_preamble_after_col:n
2580    }
```

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

```
2581  \cs_new_protected:Npn \@@_make_preamble_w_ii:nnnn #1 #2 #3 #4
2582    {
2583      \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2584      \tl_gclear:N \g_@@_pre_cell_tl
2585      \tl_gput_right:Nn \g_@@_array_preamble_tl
2586        {
2587          > {
```

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.

```
2588              \dim_set:Nn \l_@@_col_width_dim { #4 }
2589              \hbox_set:Nw \l_@@_cell_box
2590              \@@_cell_begin:w
2591              \cs_set_nopar:Npn \l_@@_hpos_cell_tl { #3 }
2592            }
2593          c
2594          < {
2595              \@@_cell_end:
2596              \hbox_set_end:
2597              #1
2598              \@@_adjust_size_box:
2599              \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2600            }
2601        }
```

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

```
2602      \int_gincr:N \c@jCol
2603      \@@_rec_preamble_after_col:n
2604    }


2605  \cs_new_protected:Npn \@@_special_W:
2606    {
2607      \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \l_@@_col_width_dim
2608        { \@@_warning:n { W~warning } }
2609    }
```

For `S` (of siunitx).

```
2610 \cs_new:Npn \@@_S #1 #2
2611   {
2612     \str_if_eq:nnTF { #2 } { [ }
2613       { \@@_make_preamble_S:w [ }
2614       { \@@_make_preamble_S:w [ ] { #2 } }
2615   }
2616 \cs_new_protected:Npn \@@_make_preamble_S:w [ #1 ]
2617   { \@@_make_preamble_S_i:n { #1 } }
2618 \cs_new_protected:Npn \@@_make_preamble_S_i:n #1
2619   {
2620     \IfPackageLoadedTF { siunitx }
2621       {
2622         \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2623         \tl_gclear:N \g_@@_pre_cell_tl
2624         \tl_gput_right:Nn \g_@@_array_preamble_tl
2625           {
2626             > {
2627                 \@@_cell_begin:w
2628                 \keys_set:nn { siunitx } { #1 }
2629                 \siunitx_cell_begin:w
2630               }
2631             c
2632             < { \siunitx_cell_end: \@@_cell_end: }
2633           }
```

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

```
2634         \int_gincr:N \c@jCol
2635         \@@_rec_preamble_after_col:n
2636       }
2637     { \@@_fatal:n { siunitx~not~loaded } }
2638   }
```

For `(`, `[` and `\{`.

```
2639 \cs_new:cpn { @@ _ \token_to_str:N ( } #1 #2
2640   {
2641     \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.

```
2642     \int_if_zero:nTF \c@jCol
2643       {
2644         \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2645           {
```

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

```
2646             \tl_gset:Nn \g_@@_left_delim_tl { #1 }
2647             \tl_gset_eq:NN \g_@@_right_delim_tl \c_@@_dot_tl
2648             \@@_rec_preamble:n #2
2649           }
2650           {
2651             \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2652             \@@_make_preamble_iv:nn { #1 } { #2 }
2653           }
2654       }
2655     { \@@_make_preamble_iv:nn { #1 } { #2 } }
2656   }
2657 \cs_set_eq:cc { @@ _ \token_to_str:N [ } { @@ _ \token_to_str:N ( }
2658 \cs_set_eq:cc { @@ _ \token_to_str:N \{ } { @@ _ \token_to_str:N ( }
2659 \cs_new_protected:Npn \@@_make_preamble_iv:nn #1 #2
2660   {
2661     \tl_gput_right:Ne \g_@@_pre_code_after_tl
2662       { \@@_delimiter:nnn #1 { \int_eval:n { \c@jCol + 1 } } \c_true_bool }
```

69

```
2663      \tl_if_in:nnTF { ( [ \{ ) ] \} \left \right } { #2 }
2664        {
2665          \@@_error:nn { delimiter~after~opening } { #2 }
2666          \@@_rec_preamble:n
2667        }
2668        { \@@_rec_preamble:n #2 }
2669    }
```

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

```
2670  \cs_new:cpn { @@ _ \token_to_str:N \left } #1 { \use:c { @@ _ \token_to_str:N ( } }
```

For the closing delimiters. We have two arguments for the following command because we directly read the following letter in the preamble (we have to see whether we have 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}`).

```
2671  \cs_new:cpn { @@ _ \token_to_str:N ) } #1 #2
2672    {
2673      \bool_if:NT \l_@@_small_bool { \@@_fatal:n { Delimiter~with~small } }
2674      \tl_if_in:nnTF { ) ] \} } { #2 }
2675        { \@@_make_preamble_v:nnn #1 #2 }
2676        {
2677          \tl_if_eq:nnTF { \stop } { #2 }
2678            {
2679              \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2680                { \tl_gset:Nn \g_@@_right_delim_tl { #1 } }
2681                {
2682                  \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2683                  \tl_gput_right:Ne \g_@@_pre_code_after_tl
2684                    { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2685                  \@@_rec_preamble:n #2
2686                }
2687            }
2688            {
2689              \tl_if_in:nnT { ( [ \{ \left } { #2 }
2690                { \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } } }
2691              \tl_gput_right:Ne \g_@@_pre_code_after_tl
2692                { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2693              \@@_rec_preamble:n #2
2694            }
2695        }
2696    }
2697  \cs_set_eq:cc { @@ _ \token_to_str:N ] } { @@ _ \token_to_str:N ) }
2698  \cs_set_eq:cc { @@ _ \token_to_str:N \} } { @@ _ \token_to_str:N ) }
2699  \cs_new_protected:Npn \@@_make_preamble_v:nnn #1 #2 #3
2700    {
2701      \tl_if_eq:nnTF { \stop } { #3 }
2702        {
2703          \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2704            {
2705              \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2706              \tl_gput_right:Ne \g_@@_pre_code_after_tl
2707                { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2708              \tl_gset:Nn \g_@@_right_delim_tl { #2 }
2709            }
2710            {
2711              \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2712              \tl_gput_right:Ne \g_@@_pre_code_after_tl
2713                { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2714              \@@_error:nn { double~closing~delimiter } { #2 }
2715            }
2716        }
2717        {
2718          \tl_gput_right:Ne \g_@@_pre_code_after_tl
```

```
2719          { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2720        \@@_error:nn { double~closing~delimiter } { #2 }
2721        \@@_rec_preamble:n #3
2722      }
2723    }

2724  \cs_new:cpn { @@ _ \token_to_str:N \right } #1
2725    { \use:c { @@ _ \token_to_str:N ) } }
```

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

```
2726  \cs_new_protected:Npn \@@_rec_preamble_after_col:n #1
2727    {
2728      \str_if_eq:nnTF { #1 } { < }
2729        \@@_rec_preamble_after_col_i:n
2730        {
2731          \str_if_eq:nnTF { #1 } { @ }
2732            \@@_rec_preamble_after_col_ii:n
2733            {
2734              \tl_if_eq:NNTF \l_@@_vlines_clist \c_@@_all_tl
2735                {
2736                  \tl_gput_right:Nn \g_@@_array_preamble_tl
2737                    { ! { \skip_horizontal:N \arrayrulewidth } }
2738                }
2739                {
2740                  \exp_args:NNe
2741                  \clist_if_in:NnT \l_@@_vlines_clist { \int_eval:n { \c@jCol + 1 } }
2742                    {
2743                      \tl_gput_right:Nn \g_@@_array_preamble_tl
2744                        { ! { \skip_horizontal:N \arrayrulewidth } }
2745                    }
2746                }
2747              \@@_rec_preamble:n { #1 }
2748            }
2749        }
2750    }
2751  \cs_new_protected:Npn \@@_rec_preamble_after_col_i:n #1
2752    {
2753      \tl_gput_right:Nn \g_@@_array_preamble_tl { < { #1 } }
2754      \@@_rec_preamble_after_col:n
2755    }
```

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.

```
2756  \cs_new_protected:Npn \@@_rec_preamble_after_col_ii:n #1
2757    {
2758      \tl_if_eq:NNTF \l_@@_vlines_clist \c_@@_all_tl
2759        {
2760          \tl_gput_right:Nn \g_@@_array_preamble_tl
2761            { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2762        }
2763        {
2764          \exp_args:NNe
2765          \clist_if_in:NnTF \l_@@_vlines_clist { \int_eval:n { \c@jCol + 1 } }
2766            {
2767              \tl_gput_right:Nn \g_@@_array_preamble_tl
2768                { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2769            }
2770            { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { #1 } } }
2771        }
2772      \@@_rec_preamble:n
2773    }
```

```
2774  \cs_new:cpn { @@ _ * } #1 #2 #3
2775    {
2776      \tl_clear:N \l_tmpa_tl
2777      \int_step_inline:nn { #2 } { \tl_put_right:Nn \l_tmpa_tl { #3 } }
2778      \exp_last_unbraced:No \@@_rec_preamble:n \l_tmpa_tl
2779    }
```

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

```
2780  \cs_new:cpn { @@ _ \token_to_str:N \NC@find } #1 { \@@_rec_preamble:n }
```

For the case of a letter `X`. This specifier may take in an optional argument (between square brackets). That's why we test whether there is a `[` after the letter `X`.

```
2781  \cs_new:Npn \@@_X #1 #2
2782    {
2783      \str_if_eq:nnTF { #2 } { [ }
2784        { \@@_make_preamble_X:w [ }
2785        { \@@_make_preamble_X:w [ ] #2 }
2786    }
2787  \cs_new_protected:Npn \@@_make_preamble_X:w [ #1 ]
2788    { \@@_make_preamble_X_i:n { #1 } }
```

`#1` is the optional argument of the `X` specifier (a list of *key-value* pairs).

The following set of keys is for the specifier `X` in the preamble of the array. Such specifier may have as keys all the keys of `{ nicematrix / p-column }` but also a key as 1, 2, 3, etc. The following set of keys will be used to retrieve that value (in the counter `\l_@@_weight_int`).

```
2789  \keys_define:nn { nicematrix / X-column }
2790    { 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`.

```
2791  \cs_new_protected:Npn \@@_make_preamble_X_i:n #1
2792    {
```

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`).

```
2793      \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`).

```
2794      \str_set:Nn \l_@@_vpos_col_str { p }
```

The integer `\l_@@_weight_int` will be the weight of the `X` column (the initial value is 1). The user may specify a different value (such as 2, 3, etc.) by putting that value in the optional argument of the specifier. The weights of the `X` columns are used in the computation of the actual width of those columns as in `tabu` (now obsolete) or `tabularray`.

```
2795      \int_zero_new:N \l_@@_weight_int
2796      \int_set_eq:NN \l_@@_weight_int \c_one_int
2797      \@@_keys_p_column:n { #1 }
```

The unknown keys are put in `\l_tmpa_tl`

```
2798      \keys_set:no { nicematrix / X-column } \l_tmpa_tl
2799      \int_compare:nNnT \l_@@_weight_int < \c_zero_int
2800        {
2801          \@@_error_or_warning:n { negative~weight }
2802          \int_set:Nn \l_@@_weight_int { - \l_@@_weight_int }
2803        }
2804      \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).

```
2805      \bool_if:NTF \l_@@_X_columns_aux_bool
2806        {
2807          \exp_args:Nne
2808          \@@_make_preamble_ii_iv:nnn
2809            { \l_@@_weight_int \l_@@_X_columns_dim }
2810            { minipage }
2811            { \@@_no_update_width: }
2812        }
2813        {
2814          \tl_gput_right:Nn \g_@@_array_preamble_tl
2815            {
2816              > {
2817                  \@@_cell_begin:w
2818                  \bool_set_true:N \l_@@_X_bool
```

You encounter a problem on 2023-03-04: for an environment with X columns, during the first compilations (which are not the definitive one), sometimes, some cells are declared empty even if they should not. That's a problem because user's instructions may use these nodes. That's why we have added the following `\NotEmpty`.

```
2819                  \NotEmpty
```

The following code will nullify the box of the cell.

```
2820                  \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2821                    { \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`.

```
2822                  \begin { minipage } { 5 cm } \arraybackslash
2823                }
2824              c
2825              < {
2826                  \end { minipage }
2827                  \@@_cell_end:
2828                }
2829            }
2830          \int_gincr:N \c@jCol
2831          \@@_rec_preamble_after_col:n
2832        }
2833    }


2834  \cs_new_protected:Npn \@@_no_update_width:
2835    {
2836      \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2837        { \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing: }
2838    }
```

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

```
2839  \cs_new_protected:Npn \@@_make_preamble_vlism:n #1
2840    {
2841      \seq_gput_right:Ne \g_@@_cols_vlism_seq
2842        { \int_eval:n { \c@jCol + 1 } }
2843      \tl_gput_right:Ne \g_@@_array_preamble_tl
2844        { \exp_not:N ! { \skip_horizontal:N \arrayrulewidth } }
2845      \@@_rec_preamble:n
2846    }
```

The token `\stop` is a marker that we have inserted to mark the end of the preamble (as provided by the final user) that we have inserted in the TeX flow.

```
2847  \cs_set_eq:cN { @@ _ \token_to_str:N \stop } \use_none:n
```

The following lines try to catch some errors (when the final user has forgotten the preamble of its environment).

```
2848 \cs_new_protected:cpn { @@ _ \token_to_str:N \hline }
2849   { \@@_fatal:n { Preamble~forgotten } }
2850 \cs_set_eq:cc { @@ _ \token_to_str:N \Hline } { @@ _ \token_to_str:N \hline }
2851 \cs_set_eq:cc { @@ _ \token_to_str:N \toprule } { @@ _ \token_to_str:N \hline }
2852 \cs_set_eq:cc { @@ _ \token_to_str:N \CodeBefore } { @@ _ \token_to_str:N \hline }
```

13   The redefinition of \multicolumn

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

```
2853 \cs_new:Npn \@@_multicolumn:nnn #1 #2 #3
2854   {
```

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

```
2855     \multispan { #1 }
2856     \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing:
2857     \begingroup
2858     \bool_if:NT \c_@@_testphase_table_bool
2859       { \tbl_update_multicolumn_cell_data:n { #1 } }
2860     \cs_set_nopar:Npn \@addamp
2861       { \legacy_if:nTF { @firstamp } { \@firstampfalse } { \@preamerr 5 } }
```

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

```
2862     \tl_gclear:N \g_@@_preamble_tl
2863     \@@_make_m_preamble:n #2 \q_stop
```

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

```
2864     \exp_args:No \@mkpream \g_@@_preamble_tl
2865     \@addtopreamble \@empty
2866     \endgroup
2867     \bool_if:NT \c_@@_testphase_table_bool
2868       { \UseTaggingSocket { tbl / colspan } { #1 } }
```

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

```
2869     \int_compare:nNnT { #1 } > \c_one_int
2870       {
2871         \seq_gput_left:Ne \g_@@_multicolumn_cells_seq
2872           { \int_use:N \c@iRow - \int_eval:n { \c@jCol + 1 } }
2873         \seq_gput_left:Nn \g_@@_multicolumn_sizes_seq { #1 }
2874         \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
2875           {
2876             {
2877               \int_if_zero:nTF \c@jCol
2878                 { \int_eval:n { \c@iRow + 1 } }
2879                 { \int_use:N \c@iRow }
2880             }
2881             { \int_eval:n { \c@jCol + 1 } }
2882             {
2883               \int_if_zero:nTF \c@jCol
2884                 { \int_eval:n { \c@iRow + 1 } }
2885                 { \int_use:N \c@iRow }
2886             }
2887             { \int_eval:n { \c@jCol + #1 } }
2888             { } % for the name of the block
```

```
2889              }
2890          }
```

We want `\cellcolor` to be available in `\multicolumn` because `\cellcolor` of colortbl is available in `\multicolumn`.

```
2891      \RenewDocumentCommand \cellcolor { O { } m }
2892        {
2893          \@@_test_color_inside:
2894          \tl_gput_right:Ne \g_@@_pre_code_before_tl
2895            {
2896              \@@_rectanglecolor [ ##1 ]
2897                { \exp_not:n { ##2 } }
2898                { \int_use:N \c@iRow - \int_use:N \c@jCol }
2899                { \int_use:N \c@iRow - \int_eval:n { \c@jCol + #1 } }
2900            }
2901          \ignorespaces
2902        }
```

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

```
2903      \cs_set_nopar:Npn \@sharp { #3 }
2904      \@arstrut
2905      \@preamble
2906      \null
```

We add some lines.

```
2907      \int_gadd:Nn \c@jCol { #1 - 1 }
2908      \int_compare:nNnT \c@jCol > \g_@@_col_total_int
2909        { \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
2910      \ignorespaces
2911    }
```

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`.

```
2912  \cs_new_protected:Npn \@@_make_m_preamble:n #1
2913    {
2914      \str_case:nnF { #1 }
2915        {
2916          c { \@@_make_m_preamble_i:n #1 }
2917          l { \@@_make_m_preamble_i:n #1 }
2918          r { \@@_make_m_preamble_i:n #1 }
2919          > { \@@_make_m_preamble_ii:nn #1 }
2920          ! { \@@_make_m_preamble_ii:nn #1 }
2921          @ { \@@_make_m_preamble_ii:nn #1 }
2922          | { \@@_make_m_preamble_iii:n #1 }
2923          p { \@@_make_m_preamble_iv:nnn t #1 }
2924          m { \@@_make_m_preamble_iv:nnn c #1 }
2925          b { \@@_make_m_preamble_iv:nnn b #1 }
2926          w { \@@_make_m_preamble_v:nnnn { } #1 }
2927          W { \@@_make_m_preamble_v:nnnn { \@@_special_W: } #1 }
2928          \q_stop { }
2929        }
2930        {
2931          \cs_if_exist:cTF { NC @ find @ #1 }
2932            {
2933              \tl_set_eq:Nc \l_tmpa_tl { NC @ rewrite @ #1 }
2934              \exp_last_unbraced:No \@@_make_m_preamble:n \l_tmpa_tl
2935            }
2936            {
2937              \tl_if_eq:nnT { #1 } { S }
2938                { \@@_fatal:n { unknown~column~type~S } }
2939                { \@@_fatal:nn { unknown~column~type } { #1 } }
2940            }
```

```
2941          }
2942      }
```

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

```
2943  \cs_new_protected:Npn \@@_make_m_preamble_i:n #1
2944      {
2945        \tl_gput_right:Nn \g_@@_preamble_tl
2946          {
2947            > { \@@_cell_begin:w \cs_set_nopar:Npn \l_@@_hpos_cell_tl { #1 } }
2948            #1
2949            < \@@_cell_end:
2950          }
```

We test for the presence of a `<`.

```
2951        \@@_make_m_preamble_x:n
2952      }
```

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

```
2953  \cs_new_protected:Npn \@@_make_m_preamble_ii:nn #1 #2
2954      {
2955        \tl_gput_right:Nn \g_@@_preamble_tl { #1 { #2 } }
2956        \@@_make_m_preamble:n
2957      }
```

For `|`

```
2958  \cs_new_protected:Npn \@@_make_m_preamble_iii:n #1
2959      {
2960        \tl_gput_right:Nn \g_@@_preamble_tl { #1 }
2961        \@@_make_m_preamble:n
2962      }
```

For `p`, `m` and `b`

```
2963  \cs_new_protected:Npn \@@_make_m_preamble_iv:nnn #1 #2 #3
2964      {
2965        \tl_gput_right:Nn \g_@@_preamble_tl
2966          {
2967            > {
2968                \@@_cell_begin:w
2969                \begin { minipage } [ #1 ] { \dim_eval:n { #3 } }
2970                \mode_leave_vertical:
2971                \arraybackslash
2972                \vrule height \box_ht:N \@arstrutbox depth 0 pt width 0 pt
2973              }
2974            c
2975            < {
2976                \vrule height 0 pt depth \box_dp:N \@arstrutbox width 0 pt
2977                \end { minipage }
2978                \@@_cell_end:
2979              }
2980          }
```

We test for the presence of a `<`.

```
2981        \@@_make_m_preamble_x:n
2982      }
```

For `w` and `W`

```
2983  \cs_new_protected:Npn \@@_make_m_preamble_v:nnnn #1 #2 #3 #4
2984      {
2985        \tl_gput_right:Nn \g_@@_preamble_tl
2986          {
2987            > {
2988                \dim_set:Nn \l_@@_col_width_dim { #4 }
2989                \hbox_set:Nw \l_@@_cell_box
2990                \@@_cell_begin:w
```

```
2991            \cs_set_nopar:Npn \l_@@_hpos_cell_tl { #3 }
2992          }
2993        c
2994        < {
2995            \@@_cell_end:
2996            \hbox_set_end:
2997            \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
2998            #1
2999            \@@_adjust_size_box:
3000            \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
3001          }
3002      }
```

We test for the presence of a `<`.

```
3003      \@@_make_m_preamble_x:n
3004    }
```

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

```
3005  \cs_new_protected:Npn \@@_make_m_preamble_x:n #1
3006    {
3007      \str_if_eq:nnTF { #1 } { < }
3008        \@@_make_m_preamble_ix:n
3009        { \@@_make_m_preamble:n { #1 } }
3010    }
3011  \cs_new_protected:Npn \@@_make_m_preamble_ix:n #1
3012    {
3013      \tl_gput_right:Nn \g_@@_preamble_tl { < { #1 } }
3014      \@@_make_m_preamble_x:n
3015    }
```

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`).

```
3016  \cs_new_protected:Npn \@@_put_box_in_flow:
3017    {
3018      \box_set_ht:Nn \l_tmpa_box { \box_ht:N \l_tmpa_box + \l_tmpa_dim }
3019      \box_set_dp:Nn \l_tmpa_box { \box_dp:N \l_tmpa_box + \l_tmpb_dim }
3020      \tl_if_eq:NNTF \l_@@_baseline_tl \c_@@_c_tl
3021        { \box_use_drop:N \l_tmpa_box }
3022        \@@_put_box_in_flow_i:
3023    }
```

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).

```
3024  \cs_new_protected:Npn \@@_put_box_in_flow_i:
3025    {
3026      \pgfpicture
3027        \@@_qpoint:n { row - 1 }
3028        \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3029        \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
3030        \dim_gadd:Nn \g_tmpa_dim \pgf@y
3031        \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).

```
3032        \tl_if_in:NnTF \l_@@_baseline_tl { line- }
3033          {
3034            \int_set:Nn \l_tmpa_int
3035              {
3036                \str_range:Nnn
3037                  \l_@@_baseline_tl
```

```
3038              6
3039              { \tl_count:o \l_@@_baseline_tl }
3040            }
3041          \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
3042        }
3043        {
3044          \tl_if_eq:NnTF \l_@@_baseline_tl { t }
3045            { \int_set_eq:NN \l_tmpa_int \c_one_int }
3046            {
3047              \tl_if_eq:NnTF \l_@@_baseline_tl { b }
3048                { \int_set_eq:NN \l_tmpa_int \c@iRow }
3049                { \int_set:Nn \l_tmpa_int \l_@@_baseline_tl }
3050            }
3051          \bool_lazy_or:nnT
3052            { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
3053            { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
3054            {
3055              \@@_error:n { bad~value~for~baseline }
3056              \int_set_eq:NN \l_tmpa_int \c_one_int
3057            }
3058          \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }
```
We take into account the position of the mathematical axis.
```
3059              \dim_gsub:Nn \g_tmpa_dim { \fontdimen22 \textfont2 }
3060          }
3061        \dim_gsub:Nn \g_tmpa_dim \pgf@y
```
Now, \g_tmpa_dim contains the value of the *y* translation we have to to.
```
3062      \endpgfpicture
3063      \box_move_up:nn \g_tmpa_dim { \box_use_drop:N \l_tmpa_box }
3064      \box_use_drop:N \l_tmpa_box
3065    }
```

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).
```
3066 \cs_new_protected:Npn \@@_use_arraybox_with_notes_c:
3067    {
```
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.
```
3068      \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3069        {
3070          \int_compare:nNnT \c@jCol > \c_one_int
3071            {
3072              \box_set_wd:Nn \l_@@_the_array_box
3073                { \box_wd:N \l_@@_the_array_box - \arraycolsep }
3074            }
3075        }
```
We need a {minipage} because we will insert a LaTeX list for the tabular notes (that means that a \vtop{\hsize=...} is not enough).
```
3076      \begin { minipage } [ t ] { \box_wd:N \l_@@_the_array_box }
3077      \bool_if:NT \l_@@_caption_above_bool
3078        {
3079          \tl_if_empty:NF \l_@@_caption_tl
3080            {
3081              \bool_set_false:N \g_@@_caption_finished_bool
3082              \int_gzero:N \c@tabularnote
3083              \@@_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).

```
3084              \int_compare:nNnT \g_@@_notes_caption_int > \c_zero_int
3085                {
3086                  \tl_gput_right:Ne \g_@@_aux_tl
3087                    {
3088                      \tl_set:Nn \exp_not:N \l_@@_note_in_caption_tl
3089                        { \int_use:N \g_@@_notes_caption_int }
3090                    }
3091                  \int_gzero:N \g_@@_notes_caption_int
3092                }
3093            }
3094          }
```

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

```
3095        \hbox
3096          {
3097            \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.

```
3098            \@@_create_extra_nodes:
3099            \seq_if_empty:NF \g_@@_blocks_seq \@@_draw_blocks:
3100          }
```

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

```
3101      \bool_lazy_any:nT
3102        {
3103          { ! \seq_if_empty_p:N \g_@@_notes_seq }
3104          { ! \seq_if_empty_p:N \g_@@_notes_in_caption_seq }
3105          { ! \tl_if_empty_p:o \g_@@_tabularnote_tl }
3106        }
3107        \@@_insert_tabularnotes:
3108      \cs_set_eq:NN \tabularnote \@@_tabularnote_error:n
3109      \bool_if:NF \l_@@_caption_above_bool \@@_insert_caption:
3110      \end { minipage }
3111    }
```

```
3112 \cs_new_protected:Npn \@@_insert_caption:
3113   {
3114     \tl_if_empty:NF \l_@@_caption_tl
3115       {
3116         \cs_if_exist:NTF \@captype
3117           { \@@_insert_caption_i: }
3118           { \@@_error:n { caption~outside~float } }
3119       }
3120   }
```

```
3121 \cs_new_protected:Npn \@@_insert_caption_i:
3122   {
3123     \group_begin:
```

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

```
3124      \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.

```
3125        \IfPackageLoadedTF { floatrow }
3126          { \cs_set_eq:NN \@makecaption \FR@makecaption }
3127          { }
3128        \tl_if_empty:NTF \l_@@_short_caption_tl
3129          { \caption }
3130          { \caption [ \l_@@_short_caption_tl ] }
3131          { \l_@@_caption_tl }
```

In some circonstancies (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.

```
3132        \bool_if:NF \g_@@_caption_finished_bool
3133          {
3134            \bool_gset_true:N \g_@@_caption_finished_bool
3135            \int_gset_eq:NN \g_@@_notes_caption_int \c@tabularnote
3136            \int_gzero:N \c@tabularnote
3137          }
3138        \tl_if_empty:NF \l_@@_label_tl { \label { \l_@@_label_tl } }
3139        \group_end:
3140      }
3141    \cs_new_protected:Npn \@@_tabularnote_error:n #1
3142      {
3143        \@@_error_or_warning:n { tabularnote~below~the~tabular }
3144        \@@_gredirect_none:n { tabularnote~below~the~tabular }
3145      }
3146    \cs_new_protected:Npn \@@_insert_tabularnotes:
3147      {
3148        \seq_gconcat:NNN \g_@@_notes_seq \g_@@_notes_in_caption_seq \g_@@_notes_seq
3149        \int_set:Nn \c@tabularnote { \seq_count:N \g_@@_notes_seq }
3150        \skip_vertical:N 0.65ex
```

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

```
3151        \group_begin:
3152        \l_@@_notes_code_before_tl
3153        \tl_if_empty:NF \g_@@_tabularnote_tl
3154          {
3155            \g_@@_tabularnote_tl \par
3156            \tl_gclear:N \g_@@_tabularnote_tl
3157          }
```

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.

```
3158        \int_compare:nNnT \c@tabularnote > \c_zero_int
3159          {
3160            \bool_if:NTF \l_@@_notes_para_bool
3161              {
3162                \begin { tabularnotes* }
3163                  \seq_map_inline:Nn \g_@@_notes_seq
3164                    { \@@_one_tabularnote:nn ##1 }
3165                  \strut
3166                \end { tabularnotes* }
```

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

```
3167                \par
3168              }
3169              {
3170                \tabularnotes
```

```
3171                \seq_map_inline:Nn \g_@@_notes_seq
3172                  { \@@_one_tabularnote:nn ##1 }
3173                \strut
3174              \endtabularnotes
3175            }
3176        }
3177      \unskip
3178      \group_end:
3179      \bool_if:NT \l_@@_notes_bottomrule_bool
3180        {
3181          \IfPackageLoadedTF { booktabs }
3182            {
```

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

```
3183              \skip_vertical:N \aboverulesep
```

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

```
3184              { \CT@arc@ \hrule height \heavyrulewidth }
3185            }
3186            { \@@_error_or_warning:n { bottomrule~without~booktabs } }
3187        }
3188      \l_@@_notes_code_after_tl
3189      \seq_gclear:N \g_@@_notes_seq
3190      \seq_gclear:N \g_@@_notes_in_caption_seq
3191      \int_gzero:N \c@tabularnote
3192    }
```

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

```
3193 \cs_set_protected:Npn \@@_one_tabularnote:nn #1
3194   {
3195     \tl_if_novalue:nTF { #1 }
3196       { \item }
3197       { \item [ \@@_notes_label_in_list:n { #1 } ] }
3198   }
```

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.

```
3199 \cs_new_protected:Npn \@@_use_arraybox_with_notes_b:
3200   {
3201     \pgfpicture
3202       \@@_qpoint:n { row - 1 }
3203       \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3204       \@@_qpoint:n { row - \int_use:N \c@iRow - base }
3205       \dim_gsub:Nn \g_tmpa_dim \pgf@y
3206     \endpgfpicture
3207     \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3208     \int_if_zero:nT \l_@@_first_row_int
3209       {
3210         \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3211         \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3212       }
3213     \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3214   }
```

Now, the general case.

```
3215 \cs_new_protected:Npn \@@_use_arraybox_with_notes:
3216   {
```

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

```
3217     \tl_if_eq:NnT \l_@@_baseline_tl { t }
3218       { \cs_set_nopar:Npn \l_@@_baseline_tl { 1 } }
```

Now, we convert the value of `\l_@@_baseline_tl` (which should represent an integer) to an integer stored in `\l_tmpa_int`.

```
3219      \pgfpicture
3220      \@@_qpoint:n { row - 1 }
3221      \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3222      \str_if_in:NnTF \l_@@_baseline_tl { line- }
3223        {
3224          \int_set:Nn \l_tmpa_int
3225            {
3226              \str_range:Nnn
3227                \l_@@_baseline_tl
3228                6
3229                { \tl_count:o \l_@@_baseline_tl }
3230            }
3231          \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
3232        }
3233        {
3234          \int_set:Nn \l_tmpa_int \l_@@_baseline_tl
3235          \bool_lazy_or:nnT
3236            { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
3237            { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
3238            {
3239              \@@_error:n { bad~value~for~baseline }
3240              \int_set:Nn \l_tmpa_int 1
3241            }
3242          \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }
3243        }
3244      \dim_gsub:Nn \g_tmpa_dim \pgf@y
3245      \endpgfpicture
3246      \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3247      \int_if_zero:nT \l_@@_first_row_int
3248        {
3249          \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3250          \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3251        }
3252      \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3253    }
```

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.

```
3254  \cs_new_protected:Npn \@@_put_box_in_flow_bis:nn #1 #2
3255    {
```

We will compute the real width of both delimiters used.

```
3256      \dim_zero_new:N \l_@@_real_left_delim_dim
3257      \dim_zero_new:N \l_@@_real_right_delim_dim
3258      \hbox_set:Nn \l_tmpb_box
3259        {
3260          \c_math_toggle_token
3261          \left #1
3262          \vcenter
3263            {
3264              \vbox_to_ht:nn
3265                { \box_ht_plus_dp:N \l_tmpa_box }
3266                { }
3267            }
3268          \right .
3269          \c_math_toggle_token
3270        }
3271      \dim_set:Nn \l_@@_real_left_delim_dim
3272        { \box_wd:N \l_tmpb_box - \nulldelimiterspace }
3273      \hbox_set:Nn \l_tmpb_box
```

```
3274        {
3275          \c_math_toggle_token
3276          \left .
3277          \vbox_to_ht:nn
3278            { \box_ht_plus_dp:N \l_tmpa_box }
3279            { }
3280          \right #2
3281          \c_math_toggle_token
3282        }
3283      \dim_set:Nn \l_@@_real_right_delim_dim
3284        { \box_wd:N \l_tmpb_box - \nulldelimiterspace }
```

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

```
3285      \skip_horizontal:N  \l_@@_left_delim_dim
3286      \skip_horizontal:N -\l_@@_real_left_delim_dim
3287      \@@_put_box_in_flow:
3288      \skip_horizontal:N \l_@@_right_delim_dim
3289      \skip_horizontal:N -\l_@@_real_right_delim_dim
3290    }
```

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).

```
3291 \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).

```
3292    {
3293      \peek_remove_spaces:n
3294        {
3295          \peek_meaning:NTF \end
3296            \@@_analyze_end:Nn
3297            {
3298              \@@_transform_preamble:
```

Here is the call to \array (we have a dedicated macro \@@_array: because of compatibility with the classes revtex4-1 and revtex4-2).

```
3299              \exp_args:No \@@_array: \g_@@_array_preamble_tl
3300            }
3301        }
3302    }
3303    {
3304      \@@_create_col_nodes:
3305      \endarray
3306    }
```

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

```
3307 \NewDocumentEnvironment { @@-light-syntax } { b }
3308    {
```

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.

```
3309      \tl_if_empty:nT { #1 } { \@@_fatal:n { empty~environment } }
3310      \tl_map_inline:nn { #1 }
3311        {
3312          \str_if_eq:nnT { ##1 } { & }
3313            { \@@_fatal:n { ampersand~in~light-syntax } }
```

```
3314        \str_if_eq:nnT { ##1 } { \\ }
3315          { \@@_fatal:n { double-backslash~in~light-syntax } } }
3316      }
```

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 catched 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`.

```
3317      \@@_light_syntax_i:w #1 \CodeAfter \q_stop
```

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

```
3318    }
```

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.

```
3319    {
3320      \@@_create_col_nodes:
3321      \endarray
3322    }

3323  \cs_new_protected:Npn \@@_light_syntax_i:w #1\CodeAfter #2\q_stop
3324    {
3325      \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).

```
3326      \seq_clear_new:N \l_@@_rows_seq
```

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

```
3327      \tl_set_rescan:Nno \l_@@_end_of_row_tl { } \l_@@_end_of_row_tl
3328      \bool_if:NTF \l_@@_light_syntax_expanded_bool
3329        \seq_set_split:Nee
3330        \seq_set_split:Non
3331        \l_@@_rows_seq \l_@@_end_of_row_tl { #1 }
```

We delete the last row if it is empty.

```
3332      \seq_pop_right:NN \l_@@_rows_seq \l_tmpa_tl
3333      \tl_if_empty:NF \l_tmpa_tl
3334        { \seq_put_right:No \l_@@_rows_seq \l_tmpa_tl }
```

If the environment uses the option `last-row` without value (i.e. without saying the number of the rows), we have now the opportunity to compute that value. We do it, and so, if the token list `\l_@@_code_for_last_row_tl` is not empty, we will use directly where it should be.

```
3335      \int_compare:nNnT \l_@@_last_row_int = { -1 }
3336        { \int_set:Nn \l_@@_last_row_int { \seq_count:N \l_@@_rows_seq } }
```

The new value of the body (that is to say after replacement of the separators of rows and columns by `\\` and `&`) of the environment will be stored in `\l_@@_new_body_tl` in order to allow the use of commands such as `\hline` or `\hdottedline` with the key `light-syntax`).

```
3337      \tl_build_begin:N \l_@@_new_body_tl
3338      \int_zero_new:N \l_@@_nb_cols_int
```

First, we treat the first row.

```
3339      \seq_pop_left:NN \l_@@_rows_seq \l_tmpa_tl
3340      \@@_line_with_light_syntax:o \l_tmpa_tl
```

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

```
3341      \seq_map_inline:Nn \l_@@_rows_seq
3342        {
3343          \tl_build_put_right:Nn \l_@@_new_body_tl { \\ }
3344          \@@_line_with_light_syntax:n { ##1 }
3345        }
3346      \tl_build_end:N \l_@@_new_body_tl
```

```
3347      \int_compare:nNnT \l_@@_last_col_int = { -1 }
3348        {
3349          \int_set:Nn \l_@@_last_col_int
3350            { \l_@@_nb_cols_int - 1 + \l_@@_first_col_int }
3351        }
```

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.

```
3352        \@@_transform_preamble:
```

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

```
3353        \exp_args:No \@@_array: \g_@@_array_preamble_tl \l_@@_new_body_tl
3354      }
3355  \cs_new_protected:Npn \@@_line_with_light_syntax:n #1
3356    {
3357      \seq_clear_new:N \l_@@_cells_seq
3358      \seq_set_split:Nnn \l_@@_cells_seq { ~ } { #1 }
3359      \int_set:Nn \l_@@_nb_cols_int
3360        {
3361          \int_max:nn
3362            \l_@@_nb_cols_int
3363            { \seq_count:N \l_@@_cells_seq }
3364        }
3365      \seq_pop_left:NN \l_@@_cells_seq \l_tmpa_tl
3366      \exp_args:NNo \tl_build_put_right:Nn \l_@@_new_body_tl \l_tmpa_tl
3367      \seq_map_inline:Nn \l_@@_cells_seq
3368        { \tl_build_put_right:Nn \l_@@_new_body_tl { & ##1 } }
3369    }
3370  \cs_generate_variant:Nn \@@_line_with_light_syntax:n { o }
```

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

```
3371  \cs_new_protected:Npn \@@_analyze_end:Nn #1 #2
3372    {
3373      \str_if_eq:onT \g_@@_name_env_str { #2 }
3374        { \@@_fatal:n { empty~environment } }
```

We reput in the stream the `\end{...}` we have extracted and the user will have an error for incorrect nested environments.

```
3375      \end { #2 }
3376    }
```

The command `\@@_create_col_nodes:` will construct a special last row. That last row is a false row used to create the `col` nodes and to fix the width of the columns (when the array is constructed with an option which specifies the width of the columns such as `columns-width`).

```
3377  \cs_new:Npn \@@_create_col_nodes:
3378    {
3379      \crcr
3380      \int_if_zero:nT \l_@@_first_col_int
3381        {
3382          \omit
3383          \hbox_overlap_left:n
3384            {
3385              \bool_if:NT \l_@@_code_before_bool
3386                { \pgfsys@markposition { \@@_env: - col - 0 } }
3387              \pgfpicture
3388              \pgfrememberpicturepositiononpagetrue
3389              \pgfcoordinate { \@@_env: - col - 0 } \pgfpointorigin
3390              \str_if_empty:NF \l_@@_name_str
```

```
3391              { \pgfnodealias { \l_@@_name_str - col - 0 } { \@@_env: - col - 0 } }
3392            \endpgfpicture
3393            \skip_horizontal:N 2\col@sep
3394            \skip_horizontal:N \g_@@_width_first_col_dim
3395          }
3396        &
3397      }
3398    \omit
```

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

```
3399        \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).

```
3400        \int_if_zero:nTF \l_@@_first_col_int
3401          {
3402            \bool_if:NT \l_@@_code_before_bool
3403              {
3404                \hbox
3405                  {
3406                    \skip_horizontal:N -0.5\arrayrulewidth
3407                    \pgfsys@markposition { \@@_env: - col - 1 }
3408                    \skip_horizontal:N 0.5\arrayrulewidth
3409                  }
3410              }
3411            \pgfpicture
3412            \pgfrememberpicturepositiononpagetrue
3413            \pgfcoordinate { \@@_env: - col - 1 }
3414              { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3415            \str_if_empty:NF \l_@@_name_str
3416              { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
3417            \endpgfpicture
3418          }
3419          {
3420            \bool_if:NT \l_@@_code_before_bool
3421              {
3422                \hbox
3423                  {
3424                    \skip_horizontal:N 0.5\arrayrulewidth
3425                    \pgfsys@markposition { \@@_env: - col - 1 }
3426                    \skip_horizontal:N -0.5\arrayrulewidth
3427                  }
3428              }
3429            \pgfpicture
3430            \pgfrememberpicturepositiononpagetrue
3431            \pgfcoordinate { \@@_env: - col - 1 }
3432              { \pgfpoint { 0.5 \arrayrulewidth } \c_zero_dim }
3433            \str_if_empty:NF \l_@@_name_str
3434              { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
3435            \endpgfpicture
3436          }
```

We compute in \g_tmpa_skip the common width of the columns (it's a skip and not a dimension). We use a global variable because we are in a cell of an \halign and because we have to use that variable in other cells (of the same row). The affectation of \g_tmpa_skip, like all the affectations, must be done after the \omit of the cell.

We give a default value for \g_tmpa_skip (0 pt plus 1 fill) but we will add some dimensions to it.

```
3437        \skip_gset:Nn \g_tmpa_skip { 0 pt~plus 1 fill }
3438        \bool_if:NF \l_@@_auto_columns_width_bool
3439          { \dim_compare:nNnT \l_@@_columns_width_dim > \c_zero_dim }
3440          {
3441            \bool_lazy_and:nnTF
3442              \l_@@_auto_columns_width_bool
```

```
3443          { \bool_not_p:n \l_@@_block_auto_columns_width_bool }
3444          { \skip_gadd:Nn \g_tmpa_skip \g_@@_max_cell_width_dim }
3445          { \skip_gadd:Nn \g_tmpa_skip \l_@@_columns_width_dim }
3446        \skip_gadd:Nn \g_tmpa_skip { 2 \col@sep }
3447      }
3448    \skip_horizontal:N \g_tmpa_skip
3449    \hbox
3450      {
3451        \bool_if:NT \l_@@_code_before_bool
3452          {
3453            \hbox
3454              {
3455                \skip_horizontal:N -0.5\arrayrulewidth
3456                \pgfsys@markposition { \@@_env: - col - 2 }
3457                \skip_horizontal:N 0.5\arrayrulewidth
3458              }
3459          }
3460        \pgfpicture
3461        \pgfrememberpicturepositiononpagetrue
3462        \pgfcoordinate { \@@_env: - col - 2 }
3463          { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3464        \str_if_empty:NF \l_@@_name_str
3465          { \pgfnodealias { \l_@@_name_str - col - 2 } { \@@_env: - col - 2 } }
3466        \endpgfpicture
3467      }
```

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.

```
3468        \int_gset_eq:NN \g_tmpa_int \c_one_int
3469        \bool_if:NTF \g_@@_last_col_found_bool
3470          { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 3 } \c_zero_int } }
3471          { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 2 } \c_zero_int } }
3472          {
3473            &
3474            \omit
3475            \int_gincr:N \g_tmpa_int
```

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

```
3476            \skip_horizontal:N \g_tmpa_skip
3477            \bool_if:NT \l_@@_code_before_bool
3478              {
3479                \hbox
3480                  {
3481                    \skip_horizontal:N -0.5\arrayrulewidth
3482                    \pgfsys@markposition
3483                      { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3484                    \skip_horizontal:N 0.5\arrayrulewidth
3485                  }
3486              }
```

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

```
3487            \pgfpicture
3488            \pgfrememberpicturepositiononpagetrue
3489            \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3490              { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3491            \str_if_empty:NF \l_@@_name_str
3492              {
3493                \pgfnodealias
3494                  { \l_@@_name_str - col - \int_eval:n { \g_tmpa_int + 1 } }
3495                  { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3496              }
3497            \endpgfpicture
3498          }
```

```
3499              &
3500              \omit
```

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

```
3501              \int_if_zero:nT \g_@@_col_total_int
3502                { \skip_gset:Nn \g_tmpa_skip { 0 pt~plus 1 fill } }
3503              \skip_horizontal:N \g_tmpa_skip
3504              \int_gincr:N \g_tmpa_int
3505              \bool_lazy_any:nF
3506                {
3507                  \g_@@_delims_bool
3508                  \l_@@_tabular_bool
3509                  { ! \clist_if_empty_p:N \l_@@_vlines_clist }
3510                  \l_@@_exterior_arraycolsep_bool
3511                  \l_@@_bar_at_end_of_pream_bool
3512                }
3513              { \skip_horizontal:N -\col@sep }
3514              \bool_if:NT \l_@@_code_before_bool
3515                {
3516                  \hbox
3517                    {
3518                      \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.

```
3519                      \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3520                        { \skip_horizontal:N -\arraycolsep }
3521                      \pgfsys@markposition
3522                        { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3523                      \skip_horizontal:N 0.5\arrayrulewidth
3524                      \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3525                        { \skip_horizontal:N \arraycolsep }
3526                    }
3527                }
3528          \pgfpicture
3529            \pgfrememberpicturepositiononpagetrue
3530            \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3531              {
3532                \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3533                  {
3534                    \pgfpoint
3535                      { - 0.5 \arrayrulewidth - \arraycolsep }
3536                      \c_zero_dim
3537                  }
3538                  { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3539              }
3540            \str_if_empty:NF \l_@@_name_str
3541              {
3542                \pgfnodealias
3543                  { \l_@@_name_str - col - \int_eval:n { \g_tmpa_int + 1 } }
3544                  { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3545              }
3546          \endpgfpicture


3547      \bool_if:NT \g_@@_last_col_found_bool
3548        {
3549          \hbox_overlap_right:n
3550            {
3551              \skip_horizontal:N \g_@@_width_last_col_dim
3552              \skip_horizontal:N \col@sep
3553              \bool_if:NT \l_@@_code_before_bool
```

```
3554               {
3555                 \pgfsys@markposition
3556                   { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3557               }
3558           \pgfpicture
3559           \pgfrememberpicturepositiononpagetrue
3560           \pgfcoordinate
3561             { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3562             \pgfpointorigin
3563           \str_if_empty:NF \l_@@_name_str
3564             {
3565               \pgfnodealias
3566                 {
3567                   \l_@@_name_str - col
3568                   - \int_eval:n { \g_@@_col_total_int + 1 }
3569                 }
3570                 { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3571             }
3572           \endpgfpicture
3573         }
3574       }
3575   % \cr
3576   }
```

Here is the preamble for the "first column" (if the user uses the key first-col)

```
3577 \tl_const:Nn \c_@@_preamble_first_col_tl
3578   {
3579     >
3580       {
```

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).

```
3581         \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
3582         \bool_gset_true:N \g_@@_after_col_zero_bool
3583         \@@_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.

```
3584         \hbox_set:Nw \l_@@_cell_box
3585         \@@_math_toggle:
3586         \@@_tuning_key_small:
```

We insert \l_@@_code_for_first_col_tl... but we don't insert it in the potential "first row" and in the potential "last row".

```
3587         \int_compare:nNnT \c@iRow > \c_zero_int
3588           {
3589             \bool_lazy_or:nnT
3590               { \int_compare_p:nNn \l_@@_last_row_int < \c_zero_int }
3591               { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
3592               {
3593                 \l_@@_code_for_first_col_tl
3594                 \xglobal \colorlet { nicematrix-first-col } { . }
3595               }
3596           }
3597       }
```

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.

```
3598     l
3599     <
3600       {
3601         \@@_math_toggle:
3602         \hbox_set_end:
```

```
3603                \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
3604                \@@_adjust_size_box:
3605                \@@_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.
```
3606                \dim_gset:Nn \g_@@_width_first_col_dim
3607                  { \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.
```
3608                \hbox_overlap_left:n
3609                  {
3610                    \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > \c_zero_dim
3611                      \@@_node_for_cell:
3612                      { \box_use_drop:N \l_@@_cell_box }
3613                    \skip_horizontal:N \l_@@_left_delim_dim
3614                    \skip_horizontal:N \l_@@_left_margin_dim
3615                    \skip_horizontal:N \l_@@_extra_left_margin_dim
3616                  }
3617                \bool_gset_false:N \g_@@_empty_cell_bool
3618                \skip_horizontal:N -2\col@sep
3619            }
3620        }
```
Here is the preamble for the "last column" (if the user uses the key last-col).
```
3621 \tl_const:Nn \c_@@_preamble_last_col_tl
3622    {
3623      >
3624        {
3625            \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).
```
3626            \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.
```
3627            \bool_gset_true:N \g_@@_last_col_found_bool
3628            \int_gincr:N \c@jCol
3629            \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.
```
3630            \hbox_set:Nw \l_@@_cell_box
3631                \@@_math_toggle:
3632                \@@_tuning_key_small:
```
We insert \l_@@_code_for_last_col_tl... but we don't insert it in the potential "first row" and in the potential "last row".
```
3633            \int_compare:nNnT \c@iRow > \c_zero_int
3634                {
3635                    \bool_lazy_or:nnT
3636                      { \int_compare_p:nNn \l_@@_last_row_int < \c_zero_int }
3637                      { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
3638                      {
3639                        \l_@@_code_for_last_col_tl
3640                        \xglobal \colorlet { nicematrix-last-col } { . }
3641                      }
3642                }
3643        }
3644      l
3645      <
3646        {
3647            \@@_math_toggle:
3648            \hbox_set_end:
3649            \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
3650            \@@_adjust_size_box:
3651            \@@_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.

```
3652            \dim_gset:Nn \g_@@_width_last_col_dim
3653               { \dim_max:nn \g_@@_width_last_col_dim { \box_wd:N \l_@@_cell_box } }
3654            \skip_horizontal:N -2\col@sep
```

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

```
3655            \hbox_overlap_right:n
3656               {
3657                 \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \c_zero_dim
3658                    {
3659                      \skip_horizontal:N \l_@@_right_delim_dim
3660                      \skip_horizontal:N \l_@@_right_margin_dim
3661                      \skip_horizontal:N \l_@@_extra_right_margin_dim
3662                      \@@_node_for_cell:
3663                    }
3664               }
3665            \bool_gset_false:N \g_@@_empty_cell_bool
3666          }
3667      }
```

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

```
3668  \NewDocumentEnvironment { NiceArray } { }
3669    {
3670      \bool_gset_false:N \g_@@_delims_bool
3671      \str_if_empty:NT \g_@@_name_env_str
3672        { \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).

```
3673      \NiceArrayWithDelims . .
3674    }
3675    { \endNiceArrayWithDelims }
```

We create the variants of the environment {NiceArrayWithDelims}.

```
3676  \cs_new_protected:Npn \@@_def_env:nnn #1 #2 #3
3677    {
3678      \NewDocumentEnvironment { #1 NiceArray } { }
3679        {
3680          \bool_gset_true:N \g_@@_delims_bool
3681          \str_if_empty:NT \g_@@_name_env_str
3682            { \str_gset:Nn \g_@@_name_env_str { #1 NiceArray } }
3683          \@@_test_if_math_mode:
3684          \NiceArrayWithDelims #2 #3
3685        }
3686        { \endNiceArrayWithDelims }
3687    }
3688  \@@_def_env:nnn p ( )
3689  \@@_def_env:nnn b [ ]
3690  \@@_def_env:nnn B \{ \}
3691  \@@_def_env:nnn v | |
3692  \@@_def_env:nnn V \| \|
```

14   The environment {NiceMatrix} and its variants

```
3693  \cs_new_protected:Npn \@@_begin_of_NiceMatrix:nn #1 #2
3694    {
3695      \bool_set_false:N \l_@@_preamble_bool
3696      \tl_clear:N \l_tmpa_tl
3697      \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3698        { \tl_set:Nn \l_tmpa_tl { @ { } } }
3699      \tl_put_right:Nn \l_tmpa_tl
3700        {
3701          *
3702            {
3703              \int_case:nnF \l_@@_last_col_int
3704                {
3705                  { -2 } { \c@MaxMatrixCols }
3706                  { -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).

```
3707                }
3708                { \int_eval:n { \l_@@_last_col_int - 1 } } }
3709            }
3710          { #2 }
3711        }
3712      \tl_set:Nn \l_tmpb_tl { \use:c { #1 NiceArray } }
3713      \exp_args:No \l_tmpb_tl \l_tmpa_tl
3714    }
3715  \cs_generate_variant:Nn \@@_begin_of_NiceMatrix:nn { n V }
3716  \clist_map_inline:nn { p , b , B , v , V }
3717    {
3718      \NewDocumentEnvironment { #1 NiceMatrix } { ! O { } }
3719        {
3720          \bool_gset_true:N \g_@@_delims_bool
3721          \str_gset:Nn \g_@@_name_env_str { #1 NiceMatrix }
3722          \int_if_zero:nT \l_@@_last_col_int
3723            {
3724              \bool_set_true:N \l_@@_last_col_without_value_bool
3725              \int_set:Nn \l_@@_last_col_int { -1 }
3726            }
3727          \keys_set:nn { nicematrix / NiceMatrix } { ##1 }
3728          \@@_begin_of_NiceMatrix:nV { #1 } \l_@@_columns_type_tl
3729        }
3730        { \use:c { end #1 NiceArray } }
3731    }
```

We define also an environment {NiceMatrix}
```
3732  \NewDocumentEnvironment { NiceMatrix } { ! O { } }
3733    {
3734      \str_gset:Nn \g_@@_name_env_str { NiceMatrix }
3735      \int_if_zero:nT \l_@@_last_col_int
3736        {
3737          \bool_set_true:N \l_@@_last_col_without_value_bool
3738          \int_set:Nn \l_@@_last_col_int { -1 }
3739        }
3740      \keys_set:nn { nicematrix / NiceMatrix } { #1 }
3741      \bool_lazy_or:nnT
3742        { \clist_if_empty_p:N \l_@@_vlines_clist }
3743        { \l_@@_except_borders_bool }
3744        { \bool_set_true:N \l_@@_NiceMatrix_without_vlines_bool }
3745      \@@_begin_of_NiceMatrix:nV { } \l_@@_columns_type_tl
3746    }
3747    { \endNiceArray }
```

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

```
3748 \cs_new_protected:Npn \@@_NotEmpty:
3749   { \bool_gset_true:N \g_@@_not_empty_cell_bool }
```

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

```
3750 \NewDocumentEnvironment { NiceTabular } { O { } m ! O { } }
3751   {
```

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.

```
3752     \dim_compare:nNnT \l_@@_width_dim = \c_zero_dim
3753       { \dim_set_eq:NN \l_@@_width_dim \linewidth }
3754     \str_gset:Nn \g_@@_name_env_str { NiceTabular }
3755     \keys_set:nn { nicematrix / NiceTabular } { #1 , #3 }
3756     \tl_if_empty:NF \l_@@_short_caption_tl
3757       {
3758         \tl_if_empty:NT \l_@@_caption_tl
3759           {
3760             \@@_error_or_warning:n { short~caption~without~caption }
3761             \tl_set_eq:NN \l_@@_caption_tl \l_@@_short_caption_tl
3762           }
3763       }
3764     \tl_if_empty:NF \l_@@_label_tl
3765       {
3766         \tl_if_empty:NT \l_@@_caption_tl
3767           { \@@_error_or_warning:n { label~without~caption } }
3768       }
3769     \NewDocumentEnvironment { TabularNote } { b }
3770       {
3771         \bool_if:NTF \l_@@_in_code_after_bool
3772           { \@@_error_or_warning:n { TabularNote~in~CodeAfter } }
3773           {
3774             \tl_if_empty:NF \g_@@_tabularnote_tl
3775               { \tl_gput_right:Nn \g_@@_tabularnote_tl { \par } }
3776             \tl_gput_right:Nn \g_@@_tabularnote_tl { ##1 }
3777           }
3778       }
3779       { }
3780     \@@_settings_for_tabular:
3781     \NiceArray { #2 }
3782   }
3783   {
3784     \endNiceArray
3785     \bool_if:NT \c_@@_testphase_table_bool
3786       { \UseTaggingSocket { tbl / hmode / end } }
3787   }
3788 \cs_new_protected:Npn \@@_settings_for_tabular:
3789   {
3790     \bool_set_true:N \l_@@_tabular_bool
3791     \cs_set_eq:NN \@@_math_toggle: \prg_do_nothing:
3792     \cs_set_eq:NN \@@_tuning_not_tabular_begin: \prg_do_nothing:
3793     \cs_set_eq:NN \@@_tuning_not_tabular_end: \prg_do_nothing:
3794   }
```

```
3795 \NewDocumentEnvironment { NiceTabularX } { m O { } m ! O { } }
3796   {
3797     \str_gset:Nn \g_@@_name_env_str { NiceTabularX }
3798     \dim_zero_new:N \l_@@_width_dim
3799     \dim_set:Nn \l_@@_width_dim { #1 }
3800     \keys_set:nn { nicematrix / NiceTabular } { #2 , #4 }
3801     \@@_settings_for_tabular:
```

```
3802      \NiceArray { #3 }
3803    }
3804    {
3805      \endNiceArray
3806      \int_if_zero:nT \g_@@_total_X_weight_int
3807        { \@@_error:n { NiceTabularX~without~X } }
3808    }


3809  \NewDocumentEnvironment { NiceTabular* } { m O { } m ! O { } }
3810    {
3811      \str_gset:Nn \g_@@_name_env_str { NiceTabular* }
3812      \dim_set:Nn \l_@@_tabular_width_dim { #1 }
3813      \keys_set:nn { nicematrix / NiceTabular } { #2 , #4 }
3814      \@@_settings_for_tabular:
3815      \NiceArray { #3 }
3816    }
3817    { \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).

```
3818  \cs_new_protected:Npn \@@_deal_with_rounded_corners:
3819    {
3820      \bool_lazy_all:nT
3821        {
3822          { \int_compare_p:nNn \l_@@_tab_rounded_corners_dim > \c_zero_dim }
3823          \l_@@_hvlines_bool
3824          { ! \g_@@_delims_bool }
3825          { ! \l_@@_except_borders_bool }
3826        }
3827        {
3828          \bool_set_true:N \l_@@_except_borders_bool
3829          \clist_if_empty:NF \l_@@_corners_clist
3830            { \@@_error:n { hvlines,~rounded-corners~and~corners } }
3831          \tl_gput_right:Nn \g_@@_pre_code_after_tl
3832            {
3833              \@@_stroke_block:nnn
3834                {
3835                  rounded-corners = \dim_use:N \l_@@_tab_rounded_corners_dim ,
3836                  draw = \l_@@_rules_color_tl
3837                }
3838                { 1-1 }
3839                { \int_use:N \c@iRow - \int_use:N \c@jCol }
3840            }
3841        }
3842    }


3843  \cs_new_protected:Npn \@@_after_array:
3844    {
```

There was a \hook_gput_code:nnn { env / tabular / begin } { nicematrix } in the command \@@_pre_array_ii: in order to come back to the standard definition of \multicolumn (in the tabulars used by the final user in the cells of our array of nicematrix) and maybe another linked to colortbl.

```
3845      \hook_gremove_code:nn { env / tabular / begin } { nicematrix }
3846      \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.

```
3847        \bool_if:NT \g_@@_last_col_found_bool
3848          { \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`.

```
3849        \bool_if:NT \l_@@_last_col_without_value_bool
3850          { \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.

```
3851        \bool_if:NT \l_@@_last_row_without_value_bool
3852          { \int_set_eq:NN \l_@@_last_row_int \g_@@_row_total_int }

3853        \tl_gput_right:Ne \g_@@_aux_tl
3854          {
3855            \seq_gset_from_clist:Nn \exp_not:N \g_@@_size_seq
3856              {
3857                \int_use:N \l_@@_first_row_int ,
3858                \int_use:N \c@iRow ,
3859                \int_use:N \g_@@_row_total_int ,
3860                \int_use:N \l_@@_first_col_int ,
3861                \int_use:N \c@jCol ,
3862                \int_use:N \g_@@_col_total_int
3863              }
3864          }
```

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`).

```
3865        \seq_if_empty:NF \g_@@_pos_of_blocks_seq
3866          {
3867            \tl_gput_right:Ne \g_@@_aux_tl
3868              {
3869                \seq_gset_from_clist:Nn \exp_not:N \g_@@_pos_of_blocks_seq
3870                  { \seq_use:Nnnn \g_@@_pos_of_blocks_seq , , , }
3871              }
3872          }
3873        \seq_if_empty:NF \g_@@_multicolumn_cells_seq
3874          {
3875            \tl_gput_right:Ne \g_@@_aux_tl
3876              {
3877                \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_cells_seq
3878                  { \seq_use:Nnnn \g_@@_multicolumn_cells_seq , , , }
3879                \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_sizes_seq
3880                  { \seq_use:Nnnn \g_@@_multicolumn_sizes_seq , , , }
3881              }
3882          }
```

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

```
3883        \@@_create_diag_nodes:
```

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

```
3884        \pgfpicture
3885        \int_step_inline:nn \c@iRow
3886          {
3887            \pgfnodealias
3888              { \@@_env: - ##1 - last }
3889              { \@@_env: - ##1 - \int_use:N \c@jCol }
```

```
3890            }
3891        \int_step_inline:nn \c@jCol
3892          {
3893            \pgfnodealias
3894              { \@@_env: - last - ##1 }
3895              { \@@_env: - \int_use:N \c@iRow - ##1 }
3896          }
3897        \str_if_empty:NF \l_@@_name_str
3898          {
3899            \int_step_inline:nn \c@iRow
3900              {
3901                \pgfnodealias
3902                  { \l_@@_name_str - ##1 - last }
3903                  { \@@_env: - ##1 - \int_use:N \c@jCol }
3904              }
3905            \int_step_inline:nn \c@jCol
3906              {
3907                \pgfnodealias
3908                  { \l_@@_name_str - last - ##1 }
3909                  { \@@_env: - \int_use:N \c@iRow - ##1 }
3910              }
3911          }
3912        \endpgfpicture
```

By default, the diagonal lines will be parallelized[11]. 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.

```
3913        \bool_if:NT \l_@@_parallelize_diags_bool
3914          {
3915            \int_gzero_new:N \g_@@_ddots_int
3916            \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.

```
3917            \dim_gzero_new:N \g_@@_delta_x_one_dim
3918            \dim_gzero_new:N \g_@@_delta_y_one_dim
3919            \dim_gzero_new:N \g_@@_delta_x_two_dim
3920            \dim_gzero_new:N \g_@@_delta_y_two_dim
3921          }
3922        \int_zero_new:N \l_@@_initial_i_int
3923        \int_zero_new:N \l_@@_initial_j_int
3924        \int_zero_new:N \l_@@_final_i_int
3925        \int_zero_new:N \l_@@_final_j_int
3926        \bool_set_false:N \l_@@_initial_open_bool
3927        \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.

```
3928        \bool_if:NT \l_@@_small_bool
3929          {
3930            \dim_set:Nn \l_@@_xdots_radius_dim { 0.7 \l_@@_xdots_radius_dim }
3931            \dim_set:Nn \l_@@_xdots_inter_dim { 0.55 \l_@@_xdots_inter_dim }
```

The dimensions \l_@@_xdots_shorten_start_dim and \l_@@_xdots_shorten_start_dim correspond to the options xdots/shorten-start and xdots/shorten-end available to the user.

```
3932            \dim_set:Nn \l_@@_xdots_shorten_start_dim
3933              { 0.6 \l_@@_xdots_shorten_start_dim }
3934            \dim_set:Nn \l_@@_xdots_shorten_end_dim
```

---

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

```
3935          { 0.6 \l_@@_xdots_shorten_end_dim }
3936        }
```

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

```
3937        \@@_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.

```
3938        \@@_compute_corners:
```

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

```
3939        \@@_adjust_pos_of_blocks_seq:

3940        \@@_deal_with_rounded_corners:
3941        \tl_if_empty:NF \l_@@_hlines_clist \@@_draw_hlines:
3942        \tl_if_empty:NF \l_@@_vlines_clist \@@_draw_vlines:
```

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

```
3943        \IfPackageLoadedTF { tikz }
3944          {
3945            \tikzset
3946              {
3947                every~picture / .style =
3948                  {
3949                    overlay ,
3950                    remember~picture ,
3951                    name~prefix = \@@_env: -
3952                  }
3953              }
3954          }
3955          { }
3956        \bool_if:NT \c_@@_tagging_array_bool
3957          { \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign: }
3958        \cs_set_eq:NN \SubMatrix \@@_SubMatrix
3959        \cs_set_eq:NN \UnderBrace \@@_UnderBrace
3960        \cs_set_eq:NN \OverBrace \@@_OverBrace
3961        \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
3962        \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
3963        \cs_set_eq:NN \line \@@_line
3964        \g_@@_pre_code_after_tl
3965        \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.

```
3966        \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).

```
3967        \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).

```
3968        % \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
3969        %   { \@@_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:`.

```
3970        \bool_set_true:N \l_@@_in_code_after_bool
3971        \exp_last_unbraced:No \@@_CodeAfter_keys: \g_nicematrix_code_after_tl
3972        \scan_stop:
3973        \tl_gclear:N \g_nicematrix_code_after_tl
3974        \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.

```
3975        \seq_if_empty:NF \g_@@_rowlistcolors_seq { \@@_clear_rowlistcolors_seq: }
3976        \tl_if_empty:NF \g_@@_pre_code_before_tl
3977          {
3978            \tl_gput_right:Ne \g_@@_aux_tl
3979              {
3980                \tl_gset:Nn \exp_not:N \g_@@_pre_code_before_tl
3981                  { \exp_not:o \g_@@_pre_code_before_tl }
3982              }
3983            \tl_gclear:N \g_@@_pre_code_before_tl
3984          }
3985        \tl_if_empty:NF \g_nicematrix_code_before_tl
3986          {
3987            \tl_gput_right:Ne \g_@@_aux_tl
3988              {
3989                \tl_gset:Nn \exp_not:N \g_@@_code_before_tl
3990                  { \exp_not:o \g_nicematrix_code_before_tl }
3991              }
3992            \tl_gclear:N \g_nicematrix_code_before_tl
3993          }

3994        \str_gclear:N \g_@@_name_env_str
3995        \@@_restore_iRow_jCol:
```

The command `\CT@arc@` contains the instruction of color for the rules of the array[12]. 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.

```
3996        \cs_gset_eq:NN \CT@arc@ \@@_old_CT@arc@
3997      }
```

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`.

```
3998    \NewDocumentCommand \@@_CodeAfter_keys: { O { } }
3999      { \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).

```
4000    \cs_new_protected:Npn \@@_adjust_pos_of_blocks_seq:
4001      {
```

---

[12]e.g. `\color[rgb]{0.5,0.5,0}`

```
4002    \seq_gset_map_x:NNn \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq
4003      { \@@_adjust_pos_of_blocks_seq_i:nnnnn ##1 }
4004    }
```

The following command must *not* be protected.

```
4005 \cs_new:Npn \@@_adjust_pos_of_blocks_seq_i:nnnnn #1 #2 #3 #4 #5
4006    {
4007      { #1 }
4008      { #2 }
4009      {
4010        \int_compare:nNnTF { #3 } > { 99 }
4011          { \int_use:N \c@iRow }
4012          { #3 }
4013      }
4014      {
4015        \int_compare:nNnTF { #4 } > { 99 }
4016          { \int_use:N \c@jCol }
4017          { #4 }
4018      }
4019      { #5 }
4020    }
```

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).

```
4021 \hook_gput_code:nnn { begindocument } { . }
4022    {
4023      \cs_new_protected:Npx \@@_draw_dotted_lines:
4024        {
4025          \c_@@_pgfortikzpicture_tl
4026          \@@_draw_dotted_lines_i:
4027          \c_@@_endpgfortikzpicture_tl
4028        }
4029    }
```

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

```
4030 \cs_new_protected:Npn \@@_draw_dotted_lines_i:
4031    {
4032      \pgfrememberpicturepositiononpagetrue
4033      \pgf@relevantforpicturesizefalse
4034      \g_@@_HVdotsfor_lines_tl
4035      \g_@@_Vdots_lines_tl
4036      \g_@@_Ddots_lines_tl
4037      \g_@@_Iddots_lines_tl
4038      \g_@@_Cdots_lines_tl
4039      \g_@@_Ldots_lines_tl
4040    }
```


```
4041 \cs_new_protected:Npn \@@_restore_iRow_jCol:
4042    {
4043      \cs_if_exist:NT \theiRow { \int_gset_eq:NN \c@iRow \l_@@_old_iRow_int }
4044      \cs_if_exist:NT \thejCol { \int_gset_eq:NN \c@jCol \l_@@_old_jCol_int }
4045    }
```

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

```
4046 \pgfdeclareshape { @@_diag_node }
4047    {
4048      \savedanchor { \five }
4049        {
4050          \dim_gset_eq:NN \pgf@x \l_tmpa_dim
```

99

```
4051          \dim_gset_eq:NN \pgf@y \l_tmpb_dim
4052        }
4053     \anchor { 5 } { \five }
4054     \anchor { center } { \pgfpointorigin }
4055     \anchor { 1 } { \five \pgf@x = 0.2 \pgf@x \pgf@y = 0.2 \pgf@y }
4056     \anchor { 2 } { \five \pgf@x = 0.4 \pgf@x \pgf@y = 0.4 \pgf@y }
4057     \anchor { 3 } { \five \pgf@x = 0.6 \pgf@x \pgf@y = 0.6 \pgf@y }
4058     \anchor { 4 } { \five \pgf@x = 0.8 \pgf@x \pgf@y = 0.8 \pgf@y }
4059     \anchor { 6 } { \five \pgf@x = 1.2 \pgf@x \pgf@y = 1.2 \pgf@y }
4060     \anchor { 7 } { \five \pgf@x = 1.4 \pgf@x \pgf@y = 1.4 \pgf@y }
4061     \anchor { 8 } { \five \pgf@x = 1.6 \pgf@x \pgf@y = 1.6 \pgf@y }
4062     \anchor { 9 } { \five \pgf@x = 1.8 \pgf@x \pgf@y = 1.8 \pgf@y }
4063   }
```

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).

```
4064 \cs_new_protected:Npn \@@_create_diag_nodes:
4065   {
4066     \pgfpicture
4067     \pgfrememberpicturepositiononpagetrue
4068     \int_step_inline:nn { \int_max:nn \c@iRow \c@jCol }
4069       {
4070         \@@_qpoint:n { col - \int_min:nn { ##1 } { \c@jCol + 1 } }
4071         \dim_set_eq:NN \l_tmpa_dim \pgf@x
4072         \@@_qpoint:n { row - \int_min:nn { ##1 } { \c@iRow + 1 } }
4073         \dim_set_eq:NN \l_tmpb_dim \pgf@y
4074         \@@_qpoint:n { col - \int_min:nn { ##1 + 1 } { \c@jCol + 1 } }
4075         \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
4076         \@@_qpoint:n { row - \int_min:nn { ##1 + 1 } { \c@iRow + 1 } }
4077         \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
4078         \pgftransformshift { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
```

Now, `\l_tmpa_dim` and `\l_tmpb_dim` become the width and the height of the node (of shape `@@_diag_node`) that we will construct.

```
4079         \dim_set:Nn \l_tmpa_dim { ( \l_@@_tmpc_dim - \l_tmpa_dim ) / 2 }
4080         \dim_set:Nn \l_tmpb_dim { ( \l_@@_tmpd_dim - \l_tmpb_dim ) / 2 }
4081         \pgfnode { @@_diag_node } { center } { } { \@@_env: - ##1 } { }
4082         \str_if_empty:NF \l_@@_name_str
4083           { \pgfnodealias { \l_@@_name_str - ##1 } { \@@_env: - ##1 } }
4084       }
```

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

```
4085     \int_set:Nn \l_tmpa_int { \int_max:nn \c@iRow \c@jCol + 1 }
4086     \@@_qpoint:n { row - \int_min:nn { \l_tmpa_int } { \c@iRow + 1 } }
4087     \dim_set_eq:NN \l_tmpa_dim \pgf@y
4088     \@@_qpoint:n { col - \int_min:nn { \l_tmpa_int } { \c@jCol + 1 } }
4089     \pgfcoordinate
4090       { \@@_env: - \int_use:N \l_tmpa_int } { \pgfpoint \pgf@x \l_tmpa_dim }
4091     \pgfnodealias
4092       { \@@_env: - last }
4093       { \@@_env: - \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
4094     \str_if_empty:NF \l_@@_name_str
4095       {
4096         \pgfnodealias
4097           { \l_@@_name_str - \int_use:N \l_tmpa_int }
4098           { \@@_env: - \int_use:N \l_tmpa_int }
4099         \pgfnodealias
4100           { \l_@@_name_str - last }
4101           { \@@_env: - last }
4102       }
4103     \endpgfpicture
4104   }
```

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.

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

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.

```
4105 \cs_new_protected:Npn \@@_find_extremities_of_line:nnnn #1 #2 #3 #4
4106   {
```

First, we declare the current cell as "dotted" because we forbide intersections of dotted lines.

```
4107     \cs_set:cpn { @@ _ dotted _ #1 - #2 } { }
```

Initialization of variables.

```
4108     \int_set:Nn \l_@@_initial_i_int { #1 }
4109     \int_set:Nn \l_@@_initial_j_int { #2 }
4110     \int_set:Nn \l_@@_final_i_int { #1 }
4111     \int_set:Nn \l_@@_final_j_int { #2 }
```

We will do two loops: one when determinating the initial cell and the other when determinating 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.

```
4112     \bool_set_false:N \l_@@_stop_loop_bool
4113     \bool_do_until:Nn \l_@@_stop_loop_bool
4114       {
4115         \int_add:Nn \l_@@_final_i_int { #3 }
4116         \int_add:Nn \l_@@_final_j_int { #4 }
```

We test if we are still in the matrix.

```
4117         \bool_set_false:N \l_@@_final_open_bool
4118         \int_compare:nNnTF \l_@@_final_i_int > \l_@@_row_max_int
4119           {
4120             \int_compare:nNnTF { #3 } = \c_one_int
4121               { \bool_set_true:N \l_@@_final_open_bool }
4122               {
4123                 \int_compare:nNnT \l_@@_final_j_int > \l_@@_col_max_int
4124                   { \bool_set_true:N \l_@@_final_open_bool }
4125               }
4126           }
4127           {
4128             \int_compare:nNnTF \l_@@_final_j_int < \l_@@_col_min_int
```

```
4129                    {
4130                      \int_compare:nNnT { #4 } = { -1 }
4131                        { \bool_set_true:N \l_@@_final_open_bool }
4132                    }
4133                    {
4134                      \int_compare:nNnT \l_@@_final_j_int > \l_@@_col_max_int
4135                        {
4136                          \int_compare:nNnT { #4 } = \c_one_int
4137                            { \bool_set_true:N \l_@@_final_open_bool }
4138                        }
4139                    }
4140                  }
4141              \bool_if:NTF \l_@@_final_open_bool
```

If we are outside the matrix, we have found the extremity of the dotted line and it's an *open* extremity.

```
4142                  {
```

We do a step backwards.

```
4143                      \int_sub:Nn \l_@@_final_i_int { #3 }
4144                      \int_sub:Nn \l_@@_final_j_int { #4 }
4145                      \bool_set_true:N \l_@@_stop_loop_bool
4146                  }
```

If we are in the matrix, we test whether the cell is empty. If it's not the case, we stop the loop because we have found the correct values for \l_@@_final_i_int and \l_@@_final_j_int.

```
4147                  {
4148                    \cs_if_exist:cTF
4149                      {
4150                        @@ _ dotted _
4151                        \int_use:N \l_@@_final_i_int -
4152                        \int_use:N \l_@@_final_j_int
4153                      }
4154                      {
4155                        \int_sub:Nn \l_@@_final_i_int { #3 }
4156                        \int_sub:Nn \l_@@_final_j_int { #4 }
4157                        \bool_set_true:N \l_@@_final_open_bool
4158                        \bool_set_true:N \l_@@_stop_loop_bool
4159                      }
4160                      {
4161                        \cs_if_exist:cTF
4162                          {
4163                            pgf @ sh @ ns @ \@@_env:
4164                            - \int_use:N \l_@@_final_i_int
4165                            - \int_use:N \l_@@_final_j_int
4166                          }
4167                          { \bool_set_true:N \l_@@_stop_loop_bool }
```

If the case is empty, we declare that the cell as non-empty. Indeed, we will draw a dotted line and the cell will be on that dotted line. All the cells of a dotted line have to be marked as "dotted" because we don't want intersections between dotted lines. We recall that the research of the extremities of the lines are all done in the same TeX group (the group of the environment), even though, when the extremities are found, each line is drawn in a TeX group that we will open for the options of the line.

```
4168                          {
4169                            \cs_set:cpn
4170                              {
4171                                @@ _ dotted _
4172                                \int_use:N \l_@@_final_i_int -
4173                                \int_use:N \l_@@_final_j_int
4174                              }
4175                              { }
4176                          }
4177                      }
4178                  }
4179              }
```

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

```
4180        \bool_set_false:N \l_@@_stop_loop_bool
4181        \bool_do_until:Nn \l_@@_stop_loop_bool
4182          {
4183            \int_sub:Nn \l_@@_initial_i_int { #3 }
4184            \int_sub:Nn \l_@@_initial_j_int { #4 }
4185            \bool_set_false:N \l_@@_initial_open_bool
4186            \int_compare:nNnTF \l_@@_initial_i_int < \l_@@_row_min_int
4187              {
4188                \int_compare:nNnTF { #3 } = \c_one_int
4189                  { \bool_set_true:N \l_@@_initial_open_bool }
4190                  {
4191                    \int_compare:nNnT \l_@@_initial_j_int = { \l_@@_col_min_int - 1 }
4192                      { \bool_set_true:N \l_@@_initial_open_bool }
4193                  }
4194              }
4195              {
4196                \int_compare:nNnTF \l_@@_initial_j_int < \l_@@_col_min_int
4197                  {
4198                    \int_compare:nNnT { #4 } = \c_one_int
4199                      { \bool_set_true:N \l_@@_initial_open_bool }
4200                  }
4201                  {
4202                    \int_compare:nNnT \l_@@_initial_j_int > \l_@@_col_max_int
4203                      {
4204                        \int_compare:nNnT { #4 } = { -1 }
4205                          { \bool_set_true:N \l_@@_initial_open_bool }
4206                      }
4207                  }
4208              }
4209            \bool_if:NTF \l_@@_initial_open_bool
4210              {
4211                \int_add:Nn \l_@@_initial_i_int { #3 }
4212                \int_add:Nn \l_@@_initial_j_int { #4 }
4213                \bool_set_true:N \l_@@_stop_loop_bool
4214              }
4215              {
4216                \cs_if_exist:cTF
4217                  {
4218                    @@ _ dotted _
4219                    \int_use:N \l_@@_initial_i_int -
4220                    \int_use:N \l_@@_initial_j_int
4221                  }
4222                  {
4223                    \int_add:Nn \l_@@_initial_i_int { #3 }
4224                    \int_add:Nn \l_@@_initial_j_int { #4 }
4225                    \bool_set_true:N \l_@@_initial_open_bool
4226                    \bool_set_true:N \l_@@_stop_loop_bool
4227                  }
4228                  {
4229                    \cs_if_exist:cTF
4230                      {
4231                        pgf @ sh @ ns @ \@@_env:
4232                        - \int_use:N \l_@@_initial_i_int
4233                        - \int_use:N \l_@@_initial_j_int
4234                      }
4235                      { \bool_set_true:N \l_@@_stop_loop_bool }
4236                      {
4237                        \cs_set:cpn
4238                          {
4239                            @@ _ dotted _
4240                            \int_use:N \l_@@_initial_i_int -
```

```
4241                        \int_use:N \l_@@_initial_j_int
4242                    }
4243                    { }
4244                }
4245            }
4246        }
4247    }
```

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.

```
4248        \seq_gput_right:Ne \g_@@_pos_of_xdots_seq
4249        {
4250            { \int_use:N \l_@@_initial_i_int }
```

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

```
4251            { \int_min:nn \l_@@_initial_j_int \l_@@_final_j_int }
4252            { \int_use:N \l_@@_final_i_int }
4253            { \int_max:nn \l_@@_initial_j_int \l_@@_final_j_int }
4254            { } % for the name of the block
4255        }
4256    }
```

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.

```
4257 \cs_new_protected:Npn \@@_open_shorten:
4258    {
4259        \bool_if:NT \l_@@_initial_open_bool
4260            { \dim_zero:N \l_@@_xdots_shorten_start_dim }
4261        \bool_if:NT \l_@@_final_open_bool
4262            { \dim_zero:N \l_@@_xdots_shorten_end_dim }
4263    }
```

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

```
4264 \cs_new_protected:Npn \@@_adjust_to_submatrix:nn #1 #2
4265    {
4266        \int_set:Nn \l_@@_row_min_int 1
4267        \int_set:Nn \l_@@_col_min_int 1
4268        \int_set_eq:NN \l_@@_row_max_int \c@iRow
4269        \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`.

```
4270        \seq_map_inline:Nn \g_@@_submatrix_seq
4271            { \@@_adjust_to_submatrix:nnnnnn { #1 } { #2 } ##1 }
4272    }
```

#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.

```
4273 \cs_set_protected:Npn \@@_adjust_to_submatrix:nnnnnn #1 #2 #3 #4 #5 #6
4274    {
4275        \int_compare:nNnF { #3 } > { #1 }
4276            {
4277                \int_compare:nNnF { #1 } > {  #5 }
4278                    {
```

```
4279            \int_compare:nNnF { #4 } > { #2 }
4280              {
4281                \int_compare:nNnF { #2 } > { #6 }
4282                  {
4283                    \int_set:Nn \l_@@_row_min_int
4284                      { \int_max:nn \l_@@_row_min_int { #3 } }
4285                    \int_set:Nn \l_@@_col_min_int
4286                      { \int_max:nn \l_@@_col_min_int { #4 } }
4287                    \int_set:Nn \l_@@_row_max_int
4288                      { \int_min:nn \l_@@_row_max_int { #5 } }
4289                    \int_set:Nn \l_@@_col_max_int
4290                      { \int_min:nn \l_@@_col_max_int { #6 } }
4291                  }
4292              }
4293          }
4294      }
4295  }

4296 \cs_new_protected:Npn \@@_set_initial_coords:
4297  {
4298    \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4299    \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4300  }
4301 \cs_new_protected:Npn \@@_set_final_coords:
4302  {
4303    \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4304    \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4305  }
4306 \cs_new_protected:Npn \@@_set_initial_coords_from_anchor:n #1
4307  {
4308    \pgfpointanchor
4309      {
4310        \@@_env:
4311        - \int_use:N \l_@@_initial_i_int
4312        - \int_use:N \l_@@_initial_j_int
4313      }
4314      { #1 }
4315    \@@_set_initial_coords:
4316  }
4317 \cs_new_protected:Npn \@@_set_final_coords_from_anchor:n #1
4318  {
4319    \pgfpointanchor
4320      {
4321        \@@_env:
4322        - \int_use:N \l_@@_final_i_int
4323        - \int_use:N \l_@@_final_j_int
4324      }
4325      { #1 }
4326    \@@_set_final_coords:
4327  }
4328 \cs_new_protected:Npn \@@_open_x_initial_dim:
4329  {
4330    \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
4331    \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
4332      {
4333        \cs_if_exist:cT
4334        { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4335          {
4336            \pgfpointanchor
4337              { \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4338              { west }
4339            \dim_set:Nn \l_@@_x_initial_dim
4340              { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
```

```
4341                      }
4342                  }
```

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

```
4343        \dim_compare:nNnT \l_@@_x_initial_dim = \c_max_dim
4344            {
4345              \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4346              \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4347              \dim_add:Nn \l_@@_x_initial_dim \col@sep
4348            }
4349      }
4350  \cs_new_protected:Npn \@@_open_x_final_dim:
4351      {
4352        \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
4353        \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
4354            {
4355              \cs_if_exist:cT
4356                { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4357                {
4358                  \pgfpointanchor
4359                    { \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4360                    { east }
4361                  \dim_set:Nn \l_@@_x_final_dim
4362                    { \dim_max:nn \l_@@_x_final_dim \pgf@x }
4363                }
4364            }
```

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

```
4365        \dim_compare:nNnT \l_@@_x_final_dim = { - \c_max_dim }
4366            {
4367              \@@_qpoint:n { col - \int_eval:n { \l_@@_final_j_int + 1 } }
4368              \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4369              \dim_sub:Nn \l_@@_x_final_dim \col@sep
4370            }
4371      }
```

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.

```
4372  \cs_new_protected:Npn \@@_draw_Ldots:nnn #1 #2 #3
4373      {
4374        \@@_adjust_to_submatrix:nn { #1 } { #2 }
4375        \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4376            {
4377              \@@_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.

```
4378              \group_begin:
4379                \@@_open_shorten:
4380                \int_if_zero:nTF { #1 }
4381                  { \color { nicematrix-first-row } }
4382                  {
```

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.

```
4383                    \int_compare:nNnT { #1 } = \l_@@_last_row_int
4384                      { \color { nicematrix-last-row } }
4385                  }
4386                \keys_set:nn { nicematrix / xdots } { #3 }
4387                \tl_if_empty:oF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4388                \@@_actually_draw_Ldots:
4389              \group_end:
4390            }
4391      }
```

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`.

```
4392  \cs_new_protected:Npn \@@_actually_draw_Ldots:
4393    {
4394      \bool_if:NTF \l_@@_initial_open_bool
4395        {
4396          \@@_open_x_initial_dim:
4397          \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4398          \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4399        }
4400        { \@@_set_initial_coords_from_anchor:n { base~east } }
4401      \bool_if:NTF \l_@@_final_open_bool
4402        {
4403          \@@_open_x_final_dim:
4404          \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4405          \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4406        }
4407        { \@@_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.

```
4408      \bool_lazy_all:nTF
4409        {
4410          \l_@@_initial_open_bool
4411          \l_@@_final_open_bool
4412          { \int_compare_p:nNn \l_@@_initial_i_int = \l_@@_last_row_int }
4413        }
4414        {
4415          \dim_add:Nn \l_@@_y_initial_dim \c_@@_shift_Ldots_last_row_dim
4416          \dim_add:Nn \l_@@_y_final_dim \c_@@_shift_Ldots_last_row_dim
4417        }
```

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

```
4418        {
4419          \dim_add:Nn \l_@@_y_initial_dim \l_@@_xdots_radius_dim
4420          \dim_add:Nn \l_@@_y_final_dim \l_@@_xdots_radius_dim
4421        }
4422      \@@_draw_line:
4423    }
```

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.

```
4424  \cs_new_protected:Npn \@@_draw_Cdots:nnn #1 #2 #3
4425    {
4426      \@@_adjust_to_submatrix:nn { #1 } { #2 }
4427      \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4428        {
4429          \@@_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.

```
4430        \group_begin:
4431          \@@_open_shorten:
4432          \int_if_zero:nTF { #1 }
4433            { \color { nicematrix-first-row } }
4434            {
```

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.

```
4435              \int_compare:nNnT { #1 } = \l_@@_last_row_int
4436                { \color { nicematrix-last-row } }
4437            }
4438          \keys_set:nn { nicematrix / xdots } { #3 }
4439          \tl_if_empty:oF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4440          \@@_actually_draw_Cdots:
4441        \group_end:
4442      }
4443    }
```

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`.

```
4444  \cs_new_protected:Npn \@@_actually_draw_Cdots:
4445    {
4446      \bool_if:NTF \l_@@_initial_open_bool
4447        { \@@_open_x_initial_dim: }
4448        { \@@_set_initial_coords_from_anchor:n { mid~east } }
4449      \bool_if:NTF \l_@@_final_open_bool
4450        { \@@_open_x_final_dim: }
4451        { \@@_set_final_coords_from_anchor:n { mid~west } }
4452      \bool_lazy_and:nnTF
4453        \l_@@_initial_open_bool
4454        \l_@@_final_open_bool
4455        {
4456          \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int }
4457          \dim_set_eq:NN \l_tmpa_dim \pgf@y
4458          \@@_qpoint:n { row - \int_eval:n { \l_@@_initial_i_int + 1 } }
4459          \dim_set:Nn \l_@@_y_initial_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
4460          \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
4461        }
4462        {
4463          \bool_if:NT \l_@@_initial_open_bool
4464            { \dim_set_eq:NN \l_@@_y_initial_dim \l_@@_y_final_dim }
4465          \bool_if:NT \l_@@_final_open_bool
4466            { \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim }
4467        }
4468      \@@_draw_line:
4469    }
4470  \cs_new_protected:Npn \@@_open_y_initial_dim:
4471    {
4472      \dim_set:Nn \l_@@_y_initial_dim { - \c_max_dim }
4473      \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
4474        {
```

```
4475        \cs_if_exist:cT
4476          { pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4477          {
4478            \pgfpointanchor
4479              { \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4480              { north }
4481            \dim_set:Nn \l_@@_y_initial_dim
4482              { \dim_max:nn \l_@@_y_initial_dim \pgf@y }
4483          }
4484        }
4485      \dim_compare:nNnT \l_@@_y_initial_dim = { - \c_max_dim }
4486        {
4487          \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4488          \dim_set:Nn \l_@@_y_initial_dim
4489            {
4490              \fp_to_dim:n
4491                {
4492                  \pgf@y
4493                  + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
4494                }
4495            }
4496        }
4497    }
4498  \cs_new_protected:Npn \@@_open_y_final_dim:
4499    {
4500      \dim_set_eq:NN \l_@@_y_final_dim \c_max_dim
4501      \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
4502        {
4503          \cs_if_exist:cT
4504            { pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4505            {
4506              \pgfpointanchor
4507                { \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4508                { south }
4509              \dim_set:Nn \l_@@_y_final_dim
4510                { \dim_min:nn \l_@@_y_final_dim \pgf@y }
4511            }
4512        }
4513      \dim_compare:nNnT \l_@@_y_final_dim = \c_max_dim
4514        {
4515          \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4516          \dim_set:Nn \l_@@_y_final_dim
4517            { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
4518        }
4519    }
```

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.

```
4520  \cs_new_protected:Npn \@@_draw_Vdots:nnn #1 #2 #3
4521    {
4522      \@@_adjust_to_submatrix:nn { #1 } { #2 }
4523      \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4524        {
4525          \@@_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.

```
4526          \group_begin:
4527            \@@_open_shorten:
4528            \int_if_zero:nTF { #2 }
4529              { \color { nicematrix-first-col } }
4530              {
4531                \int_compare:nNnT { #2 } = \l_@@_last_col_int
4532                  { \color { nicematrix-last-col } }
```

```
4533                    }
4534            \keys_set:nn { nicematrix / xdots } { #3 }
4535            \tl_if_empty:oF \l_@@_xdots_color_tl
4536              { \color { \l_@@_xdots_color_tl } }
4537            \@@_actually_draw_Vdots:
4538          \group_end:
4539        }
4540    }
```

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`.

```
4541 \cs_new_protected:Npn \@@_actually_draw_Vdots:
4542    {
```

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

```
4543      \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.

```
4544        {
4545          \@@_open_y_initial_dim:
4546          \@@_open_y_final_dim:
4547          \int_if_zero:nTF \l_@@_initial_j_int
```

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

```
4548            {
4549              \@@_qpoint:n { col - 1 }
4550              \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4551              \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
4552              \dim_sub:Nn \l_@@_x_initial_dim \l_@@_extra_left_margin_dim
4553              \dim_sub:Nn \l_@@_x_initial_dim \c_@@_shift_exterior_Vdots_dim
4554            }
4555            {
4556              \bool_lazy_and:nnTF
4557                { \int_compare_p:nNn \l_@@_last_col_int > { -2 } }
4558                { \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".

```
4559                {
4560                  \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4561                  \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4562                  \dim_add:Nn \l_@@_x_initial_dim \l_@@_right_margin_dim
4563                  \dim_add:Nn \l_@@_x_initial_dim \l_@@_extra_right_margin_dim
4564                  \dim_add:Nn \l_@@_x_initial_dim \c_@@_shift_exterior_Vdots_dim
4565                }
```

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

```
4566                {
4567                  \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4568                  \dim_set_eq:NN \l_tmpa_dim \pgf@x
4569                  \@@_qpoint:n { col - \int_eval:n { \l_@@_initial_j_int + 1 } }
4570                  \dim_set:Nn \l_@@_x_initial_dim { ( \pgf@x + \l_tmpa_dim ) / 2 }
4571                }
4572            }
4573        }
```

Now, the dotted line is *not* open on both sides (maybe open on only one side).
The boolean `\l_tmpa_bool` will indicate whether the column is of type `l` or may be considered as if.

```
4574        {
4575          \bool_set_false:N \l_tmpa_bool
4576          \bool_if:NF \l_@@_initial_open_bool
4577            {
4578              \bool_if:NF \l_@@_final_open_bool
4579                {
4580                  \@@_set_initial_coords_from_anchor:n { south~west }
4581                  \@@_set_final_coords_from_anchor:n { north~west }
4582                  \bool_set:Nn \l_tmpa_bool
4583                    { \dim_compare_p:nNn \l_@@_x_initial_dim = \l_@@_x_final_dim }
4584                }
4585            }
```

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

```
4586          \bool_if:NTF \l_@@_initial_open_bool
4587            {
4588              \@@_open_y_initial_dim:
4589              \@@_set_final_coords_from_anchor:n { north }
4590              \dim_set_eq:NN \l_@@_x_initial_dim \l_@@_x_final_dim
4591            }
4592            {
4593              \@@_set_initial_coords_from_anchor:n { south }
4594              \bool_if:NTF \l_@@_final_open_bool
4595                \@@_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.

```
4596                {
4597                  \@@_set_final_coords_from_anchor:n { north }
4598                  \dim_compare:nNnF \l_@@_x_initial_dim = \l_@@_x_final_dim
4599                    {
4600                      \dim_set:Nn \l_@@_x_initial_dim
4601                        {
4602                          \bool_if:NTF \l_tmpa_bool \dim_min:nn \dim_max:nn
4603                            \l_@@_x_initial_dim \l_@@_x_final_dim
4604                        }
4605                    }
4606                }
4607            }
4608        }
4609      \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
4610      \@@_draw_line:
4611    }
```

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.

```
4612  \cs_new_protected:Npn \@@_draw_Ddots:nnn #1 #2 #3
4613    {
4614      \@@_adjust_to_submatrix:nn { #1 } { #2 }
4615      \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4616        {
4617          \@@_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.

```
4618          \group_begin:
4619            \@@_open_shorten:
```

```
4620            \keys_set:nn { nicematrix / xdots } { #3 }
4621            \tl_if_empty:oF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4622            \@@_actually_draw_Ddots:
4623          \group_end:
4624        }
4625    }
```

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.

```
4626  \cs_new_protected:Npn \@@_actually_draw_Ddots:
4627    {
4628      \bool_if:NTF \l_@@_initial_open_bool
4629        {
4630          \@@_open_y_initial_dim:
4631          \@@_open_x_initial_dim:
4632        }
4633        { \@@_set_initial_coords_from_anchor:n { south~east } }
4634      \bool_if:NTF \l_@@_final_open_bool
4635        {
4636          \@@_open_x_final_dim:
4637          \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4638        }
4639        { \@@_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.

```
4640      \bool_if:NT \l_@@_parallelize_diags_bool
4641        {
4642          \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).

```
4643          \int_compare:nNnTF \g_@@_ddots_int = \c_one_int
```

If the diagonal line is the first one, we have no adjustment of the line to do but we store the Δ_x and the Δ_y of the line because these values will be used to draw the others diagonal lines parallels to the first one.

```
4644            {
4645              \dim_gset:Nn \g_@@_delta_x_one_dim
4646                { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4647              \dim_gset:Nn \g_@@_delta_y_one_dim
4648                { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4649            }
```

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.

```
4650            {
4651              \dim_compare:nNnF \g_@@_delta_x_one_dim = \c_zero_dim
4652                {
4653                  \dim_set:Nn \l_@@_y_final_dim
4654                    {
4655                      \l_@@_y_initial_dim +
4656                      ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4657                      \dim_ratio:nn \g_@@_delta_y_one_dim \g_@@_delta_x_one_dim
4658                    }
```

```
4659                  }
4660              }
4661          }
4662      \@@_draw_line:
4663  }
```

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.

```
4664 \cs_new_protected:Npn \@@_draw_Iddots:nnn #1 #2 #3
4665  {
4666      \@@_adjust_to_submatrix:nn { #1 } { #2 }
4667      \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4668        {
4669            \@@_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.

```
4670            \group_begin:
4671                \@@_open_shorten:
4672                \keys_set:nn { nicematrix / xdots } { #3 }
4673                \tl_if_empty:oF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4674                \@@_actually_draw_Iddots:
4675            \group_end:
4676        }
4677  }
```

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.

```
4678 \cs_new_protected:Npn \@@_actually_draw_Iddots:
4679  {
4680      \bool_if:NTF \l_@@_initial_open_bool
4681        {
4682            \@@_open_y_initial_dim:
4683            \@@_open_x_initial_dim:
4684        }
4685        { \@@_set_initial_coords_from_anchor:n { south~west } }
4686      \bool_if:NTF \l_@@_final_open_bool
4687        {
4688            \@@_open_y_final_dim:
4689            \@@_open_x_final_dim:
4690        }
4691        { \@@_set_final_coords_from_anchor:n { north~east } }
4692      \bool_if:NT \l_@@_parallelize_diags_bool
4693        {
4694            \int_gincr:N \g_@@_iddots_int
4695            \int_compare:nNnTF \g_@@_iddots_int = \c_one_int
4696              {
4697                  \dim_gset:Nn \g_@@_delta_x_two_dim
4698                    { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4699                  \dim_gset:Nn \g_@@_delta_y_two_dim
4700                    { \l_@@_y_final_dim - \l_@@_y_initial_dim }
```

```
4701                }
4702                {
4703                  \dim_compare:nNnF \g_@@_delta_x_two_dim = \c_zero_dim
4704                    {
4705                      \dim_set:Nn \l_@@_y_final_dim
4706                        {
4707                          \l_@@_y_initial_dim +
4708                          ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4709                          \dim_ratio:nn \g_@@_delta_y_two_dim \g_@@_delta_x_two_dim
4710                        }
4711                    }
4712                }
4713            }
4714        \@@_draw_line:
4715      }
```

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`

```
4716  \cs_new_protected:Npn \@@_draw_line:
4717    {
4718      \pgfrememberpicturepositiononpagetrue
4719      \pgf@relevantforpicturesizefalse
4720      \bool_lazy_or:nnTF
4721        { \tl_if_eq_p:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl }
4722        \l_@@_dotted_bool
4723        \@@_draw_standard_dotted_line:
4724        \@@_draw_unstandard_dotted_line:
4725    }
```

We have to do a special construction with `\exp_args:No` to be able to put in the list of options in the correct place in the Tikz instruction.

```
4726  \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:
4727    {
4728      \begin { scope }
4729      \@@_draw_unstandard_dotted_line:o
4730        { \l_@@_xdots_line_style_tl , \l_@@_xdots_color_tl }
4731    }
```

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 diredtly `\l_@@_xdots_color_tl`).

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

```
4732  \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:n #1
4733    {
4734      \@@_draw_unstandard_dotted_line:nooo
```

```
4735        { #1 }
4736        \l_@@_xdots_up_tl
4737        \l_@@_xdots_down_tl
4738        \l_@@_xdots_middle_tl
4739      }
4740    \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 continous line with a non-standard style.

```
4741    \hook_gput_code:nnn { begindocument } { . }
4742      {
4743        \IfPackageLoadedTF { tikz }
4744          {
4745            \tikzset
4746              {
4747                @@_node_above / .style = { sloped , above } ,
4748                @@_node_below / .style = { sloped , below } ,
4749                @@_node_middle / .style =
4750                  {
4751                    sloped ,
4752                    inner~sep = \c_@@_innersep_middle_dim
4753                  }
4754              }
4755          }
4756          { }
4757      }


4758    \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:nnnn #1 #2 #3 #4
4759      {
```

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.

```
4760        \dim_zero_new:N \l_@@_l_dim
4761        \dim_set:Nn \l_@@_l_dim
4762          {
4763            \fp_to_dim:n
4764              {
4765                sqrt
4766                  (
4767                    ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
4768                      +
4769                    ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
4770                  )
4771              }
4772          }
```

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.

```
4773        \dim_compare:nNnT \l_@@_l_dim < \c_@@_max_l_dim
4774          {
4775            \dim_compare:nNnT \l_@@_l_dim  > { 1 pt }
4776              \@@_draw_unstandard_dotted_line_i:
4777          }
```

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

```
4778        \bool_if:NT \l_@@_xdots_h_labels_bool
4779          {
```

```
4780            \tikzset
4781              {
4782                @@_node_above / .style = { auto = left } ,
4783                @@_node_below / .style = { auto = right } ,
4784                @@_node_middle / .style = { inner~sep = \c_@@_innersep_middle_dim }
4785              }
4786          }
4787        \tl_if_empty:nF { #4 }
4788          { \tikzset { @@_node_middle / .append~style = { fill = white } } }
4789        \draw
4790          [ #1 ]
4791              ( \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).

```
4792              -- node [ @@_node_middle] { $ \scriptstyle #4 $ }
4793              node [ @@_node_below ] { $ \scriptstyle #3 $ }
4794              node [ @@_node_above ] { $ \scriptstyle #2 $ }
4795              ( \l_@@_x_final_dim , \l_@@_y_final_dim ) ;
4796        \end { scope }
4797      }
4798    \cs_new_protected:Npn \@@_draw_unstandard_dotted_line_i:
4799      {
4800        \dim_set:Nn \l_tmpa_dim
4801          {
4802            \l_@@_x_initial_dim
4803            + ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
4804            * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
4805          }
4806        \dim_set:Nn \l_tmpb_dim
4807          {
4808            \l_@@_y_initial_dim
4809            + ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
4810            * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
4811          }
4812        \dim_set:Nn \l_@@_tmpc_dim
4813          {
4814            \l_@@_x_final_dim
4815            - ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
4816            * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
4817          }
4818        \dim_set:Nn \l_@@_tmpd_dim
4819          {
4820            \l_@@_y_final_dim
4821            - ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
4822            * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
4823          }
4824        \dim_set_eq:NN \l_@@_x_initial_dim \l_tmpa_dim
4825        \dim_set_eq:NN \l_@@_y_initial_dim \l_tmpb_dim
4826        \dim_set_eq:NN \l_@@_x_final_dim \l_@@_tmpc_dim
4827        \dim_set_eq:NN \l_@@_y_final_dim \l_@@_tmpd_dim
4828      }
4829    \cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:nnnn { n o o o }
```

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

```
4830    \cs_new_protected:Npn \@@_draw_standard_dotted_line:
4831      {
4832        \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.

```
4833        \dim_zero_new:N \l_@@_l_dim
4834        \dim_set:Nn \l_@@_l_dim
4835          {
4836            \fp_to_dim:n
4837              {
4838                sqrt
4839                  (
4840                    ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
4841                        +
4842                    ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
4843                  )
4844              }
4845          }
```

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.

```
4846        \dim_compare:nNnT \l_@@_l_dim < \c_@@_max_l_dim
4847          {
4848            \dim_compare:nNnT \l_@@_l_dim  > { 1 pt }
4849              \@@_draw_standard_dotted_line_i:
4850          }
4851        \group_end:
4852        \bool_lazy_all:nF
4853          {
4854            { \tl_if_empty_p:N \l_@@_xdots_up_tl }
4855            { \tl_if_empty_p:N \l_@@_xdots_down_tl }
4856            { \tl_if_empty_p:N \l_@@_xdots_middle_tl }
4857          }
4858          \l_@@_labels_standard_dotted_line:
4859      }
4860  \dim_const:Nn \c_@@_max_l_dim { 50 cm }
4861  \cs_new_protected:Npn \@@_draw_standard_dotted_line_i:
4862      {
```

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

```
4863        \int_set:Nn \l_tmpa_int
4864          {
4865            \dim_ratio:nn
4866              {
4867                \l_@@_l_dim
4868                - \l_@@_xdots_shorten_start_dim
4869                - \l_@@_xdots_shorten_end_dim
4870              }
4871              \l_@@_xdots_inter_dim
4872          }
```

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

```
4873        \dim_set:Nn \l_tmpa_dim
4874          {
4875            ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4876            \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
4877          }
4878        \dim_set:Nn \l_tmpb_dim
4879          {
4880            ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
4881            \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
4882          }
```

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

```
4883      \dim_gadd:Nn \l_@@_x_initial_dim
4884        {
4885          ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4886          \dim_ratio:nn
4887            {
4888              \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
4889              + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
4890            }
4891            { 2 \l_@@_l_dim }
4892        }
4893      \dim_gadd:Nn \l_@@_y_initial_dim
4894        {
4895          ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
4896          \dim_ratio:nn
4897            {
4898              \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
4899              + \l_@@_xdots_shorten_start_dim  - \l_@@_xdots_shorten_end_dim
4900            }
4901            { 2 \l_@@_l_dim }
4902        }
4903      \pgf@relevantforpicturesizefalse
4904      \int_step_inline:nnn \c_zero_int \l_tmpa_int
4905        {
4906          \pgfpathcircle
4907            { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
4908            { \l_@@_xdots_radius_dim }
4909          \dim_add:Nn \l_@@_x_initial_dim \l_tmpa_dim
4910          \dim_add:Nn \l_@@_y_initial_dim \l_tmpb_dim
4911        }
4912      \pgfusepathqfill
4913    }


4914  \cs_new_protected:Npn \l_@@_labels_standard_dotted_line:
4915    {
4916      \pgfscope
4917      \pgftransformshift
4918        {
4919          \pgfpointlineattime { 0.5 }
4920            { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
4921            { \pgfpoint \l_@@_x_final_dim \l_@@_y_final_dim }
4922        }
4923      \fp_set:Nn \l_tmpa_fp
4924        {
4925          atand
4926          (
4927            \l_@@_y_final_dim - \l_@@_y_initial_dim ,
4928            \l_@@_x_final_dim - \l_@@_x_initial_dim
4929          )
4930        }
4931      \pgftransformrotate { \fp_use:N \l_tmpa_fp }
4932      \bool_if:NF \l_@@_xdots_h_labels_bool { \fp_zero:N \l_tmpa_fp }
4933      \tl_if_empty:NF \l_@@_xdots_middle_tl
4934        {
4935          \begin { pgfscope }
4936          \pgfset { inner~sep = \c_@@_innersep_middle_dim }
4937          \pgfnode
4938            { rectangle }
4939            { center }
4940            {
4941              \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
```

118

```
4942                    {
4943                        \c_math_toggle_token
4944                        \scriptstyle \l_@@_xdots_middle_tl
4945                        \c_math_toggle_token
4946                    }
4947                }
4948                { }
4949                {
4950                    \pgfsetfillcolor { white }
4951                    \pgfusepath { fill }
4952                }
4953            \end { pgfscope }
4954        }
4955        \tl_if_empty:NF \l_@@_xdots_up_tl
4956            {
4957                \pgfnode
4958                    { rectangle }
4959                    { south }
4960                    {
4961                        \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4962                            {
4963                                \c_math_toggle_token
4964                                \scriptstyle \l_@@_xdots_up_tl
4965                                \c_math_toggle_token
4966                            }
4967                    }
4968                    { }
4969                    { \pgfusepath { } }
4970            }
4971        \tl_if_empty:NF \l_@@_xdots_down_tl
4972            {
4973                \pgfnode
4974                    { rectangle }
4975                    { north }
4976                    {
4977                        \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
4978                            {
4979                                \c_math_toggle_token
4980                                \scriptstyle \l_@@_xdots_down_tl
4981                                \c_math_toggle_token
4982                            }
4983                    }
4984                    { }
4985                    { \pgfusepath { } }
4986            }
4987        \endpgfscope
4988    }
```

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 thats' 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.

```
4989  \hook_gput_code:nnn { begindocument } { . }
4990    {
4991      \cs_set_nopar:Npn \l_@@_argspec_tl { m E { _ ^ : } { { } { } { } } }
4992      \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
4993      \cs_new_protected:Npn \@@_Ldots
4994        { \@@_collect_options:n { \@@_Ldots_i } }
4995      \exp_args:NNo \NewDocumentCommand \@@_Ldots_i \l_@@_argspec_tl
4996        {
4997          \int_if_zero:nTF \c@jCol
4998            { \@@_error:nn { in~first~col } \Ldots }
4999            {
5000              \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
5001                { \@@_error:nn { in~last~col } \Ldots }
5002                {
5003                  \@@_instruction_of_type:nnn \c_false_bool { Ldots }
5004                    { #1 , down = #2 , up = #3 , middle = #4 }
5005                }
5006            }
5007          \bool_if:NF \l_@@_nullify_dots_bool
5008            { \phantom { \ensuremath { \@@_old_ldots } } }
5009          \bool_gset_true:N \g_@@_empty_cell_bool
5010        }


5011      \cs_new_protected:Npn \@@_Cdots
5012        { \@@_collect_options:n { \@@_Cdots_i } }
5013      \exp_args:NNo \NewDocumentCommand \@@_Cdots_i \l_@@_argspec_tl
5014        {
5015          \int_if_zero:nTF \c@jCol
5016            { \@@_error:nn { in~first~col } \Cdots }
5017            {
5018              \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
5019                { \@@_error:nn { in~last~col } \Cdots }
5020                {
5021                  \@@_instruction_of_type:nnn \c_false_bool { Cdots }
5022                    { #1 , down = #2 , up = #3 , middle = #4 }
5023                }
5024            }
5025          \bool_if:NF \l_@@_nullify_dots_bool
5026            { \phantom { \ensuremath { \@@_old_cdots } } }
5027          \bool_gset_true:N \g_@@_empty_cell_bool
5028        }


5029      \cs_new_protected:Npn \@@_Vdots
5030        { \@@_collect_options:n { \@@_Vdots_i } }
5031      \exp_args:NNo \NewDocumentCommand \@@_Vdots_i \l_@@_argspec_tl
5032        {
5033          \int_if_zero:nTF \c@iRow
5034            { \@@_error:nn { in~first~row } \Vdots }
5035            {
5036              \int_compare:nNnTF \c@iRow = \l_@@_last_row_int
5037                { \@@_error:nn { in~last~row } \Vdots }
5038                {
5039                  \@@_instruction_of_type:nnn \c_false_bool { Vdots }
5040                    { #1 , down = #2 , up = #3 , middle = #4 }
5041                }
5042            }
5043          \bool_if:NF \l_@@_nullify_dots_bool
5044            { \phantom { \ensuremath { \@@_old_vdots } } }
5045          \bool_gset_true:N \g_@@_empty_cell_bool
5046        }
```

```
5047    \cs_new_protected:Npn \@@_Ddots
5048      { \@@_collect_options:n { \@@_Ddots_i } }
5049    \exp_args:NNo \NewDocumentCommand \@@_Ddots_i \l_@@_argspec_tl
5050      {
5051        \int_case:nnF \c@iRow
5052          {
5053            0                      { \@@_error:nn { in~first~row } \Ddots }
5054            \l_@@_last_row_int { \@@_error:nn { in~last~row } \Ddots }
5055          }
5056          {
5057            \int_case:nnF \c@jCol
5058              {
5059                0                      { \@@_error:nn { in~first~col } \Ddots }
5060                \l_@@_last_col_int { \@@_error:nn { in~last~col } \Ddots }
5061              }
5062              {
5063                \keys_set_known:nn { nicematrix / Ddots } { #1 }
5064                \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Ddots }
5065                  { #1 , down = #2 , up = #3 , middle = #4 }
5066              }
5067
5068          }
5069        \bool_if:NF \l_@@_nullify_dots_bool
5070          { \phantom { \ensuremath { \@@_old_ddots } } }
5071        \bool_gset_true:N \g_@@_empty_cell_bool
5072      }


5073    \cs_new_protected:Npn \@@_Iddots
5074      { \@@_collect_options:n { \@@_Iddots_i } }
5075    \exp_args:NNo \NewDocumentCommand \@@_Iddots_i \l_@@_argspec_tl
5076      {
5077        \int_case:nnF \c@iRow
5078          {
5079            0                      { \@@_error:nn { in~first~row } \Iddots }
5080            \l_@@_last_row_int { \@@_error:nn { in~last~row } \Iddots }
5081          }
5082          {
5083            \int_case:nnF \c@jCol
5084              {
5085                0                      { \@@_error:nn { in~first~col } \Iddots }
5086                \l_@@_last_col_int { \@@_error:nn { in~last~col } \Iddots }
5087              }
5088              {
5089                \keys_set_known:nn { nicematrix / Ddots } { #1 }
5090                \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Iddots }
5091                  { #1 , down = #2 , up = #3 , middle = #4 }
5092              }
5093          }
5094        \bool_if:NF \l_@@_nullify_dots_bool
5095          { \phantom { \ensuremath { \@@_old_iddots } } }
5096        \bool_gset_true:N \g_@@_empty_cell_bool
5097      }
5098    }
```

End of the \AddToHook.

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

```
5099  \keys_define:nn { nicematrix / Ddots }
5100    {
5101      draw-first .bool_set:N = \l_@@_draw_first_bool ,
5102      draw-first .default:n = true ,
5103      draw-first .value_forbidden:n = true
5104    }
```

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

```
5105 \cs_new_protected:Npn \@@_Hspace:
5106   {
5107     \bool_gset_true:N \g_@@_empty_cell_bool
5108     \hspace
5109   }
```

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`.

```
5110 \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`.

```
5111 \cs_new:Npn \@@_Hdotsfor:
5112   {
5113     \bool_lazy_and:nnTF
5114       { \int_if_zero_p:n \c@jCol }
5115       { \int_if_zero_p:n \l_@@_first_col_int }
5116       {
5117         \bool_if:NTF \g_@@_after_col_zero_bool
5118           {
5119             \multicolumn { 1 } { c } { }
5120             \@@_Hdotsfor_i
5121           }
5122           { \@@_fatal:n { Hdotsfor~in~col~0 } }
5123       }
5124       {
5125         \multicolumn { 1 } { c } { }
5126         \@@_Hdotsfor_i
5127       }
5128   }
```

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:`).

```
5129 \hook_gput_code:nnn { begindocument } { . }
5130   {
5131     \cs_set_nopar:Npn \l_@@_argspec_tl { m m O { } E { _ ^ : } { { } { } { } } }
5132     \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.

```
5133     \cs_new_protected:Npn \@@_Hdotsfor_i
5134       { \@@_collect_options:n { \@@_Hdotsfor_ii } }
5135     \exp_args:NNo \NewDocumentCommand \@@_Hdotsfor_ii \l_@@_argspec_tl
5136       {
5137         \tl_gput_right:Ne \g_@@_HVdotsfor_lines_tl
5138           {
5139             \@@_Hdotsfor:nnnn
5140               { \int_use:N \c@iRow }
5141               { \int_use:N \c@jCol }
5142               { #2 }
5143               {
5144                 #1 , #3 ,
5145                 down = \exp_not:n { #4 } ,
5146                 up = \exp_not:n { #5 } ,
5147                 middle = \exp_not:n { #6 }
5148               }
5149           }
5150         \prg_replicate:nn { #2 - 1 }
5151           {
```

```
5152              &
5153              \multicolumn { 1 } { c } { }
5154              \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
5155            }
5156        }
5157    }

5158  \cs_new_protected:Npn \@@_Hdotsfor:nnnn #1 #2 #3 #4
5159    {
5160      \bool_set_false:N \l_@@_initial_open_bool
5161      \bool_set_false:N \l_@@_final_open_bool
```

For the row, it's easy.

```
5162      \int_set:Nn \l_@@_initial_i_int { #1 }
5163      \int_set_eq:NN \l_@@_final_i_int \l_@@_initial_i_int
```

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

```
5164      \int_compare:nNnTF { #2 } = \c_one_int
5165        {
5166          \int_set_eq:NN \l_@@_initial_j_int \c_one_int
5167          \bool_set_true:N \l_@@_initial_open_bool
5168        }
5169        {
5170          \cs_if_exist:cTF
5171            {
5172              pgf @ sh @ ns @ \@@_env:
5173              - \int_use:N \l_@@_initial_i_int
5174              - \int_eval:n { #2 - 1 }
5175            }
5176            { \int_set:Nn \l_@@_initial_j_int { #2 - 1 } }
5177            {
5178              \int_set:Nn \l_@@_initial_j_int { #2 }
5179              \bool_set_true:N \l_@@_initial_open_bool
5180            }
5181        }
5182      \int_compare:nNnTF { #2 + #3 -1 } = \c@jCol
5183        {
5184          \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5185          \bool_set_true:N \l_@@_final_open_bool
5186        }
5187        {
5188          \cs_if_exist:cTF
5189            {
5190              pgf @ sh @ ns @ \@@_env:
5191              - \int_use:N \l_@@_final_i_int
5192              - \int_eval:n { #2 + #3 }
5193            }
5194            { \int_set:Nn \l_@@_final_j_int { #2 + #3 } }
5195            {
5196              \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5197              \bool_set_true:N \l_@@_final_open_bool
5198            }
5199        }
5200      \group_begin:
5201      \@@_open_shorten:
5202      \int_if_zero:nTF { #1 }
5203        { \color { nicematrix-first-row } }
5204        {
5205          \int_compare:nNnT { #1 } = \g_@@_row_total_int
5206            { \color { nicematrix-last-row } }
5207        }
5208
5209      \keys_set:nn { nicematrix / xdots } { #4 }
5210      \tl_if_empty:oF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
```

123

```
5211          \@@_actually_draw_Ldots:
5212          \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).

```
5213          \int_step_inline:nnn { #2 } { #2 + #3 - 1 }
5214            { \cs_set:cpn { @@ _ dotted _ #1 - ##1 } { } }
5215      }
```

```
5216  \hook_gput_code:nnn { begindocument } { . }
5217    {
5218      \cs_set_nopar:Npn \l_@@_argspec_tl { m m O { } E { _ ^ : } { { } { } { } } }
5219      \tl_set_rescan:Nno  \l_@@_argspec_tl { } \l_@@_argspec_tl
5220      \cs_new_protected:Npn \@@_Vdotsfor:
5221        { \@@_collect_options:n { \@@_Vdotsfor_i } }
5222      \exp_args:NNo \NewDocumentCommand \@@_Vdotsfor_i \l_@@_argspec_tl
5223        {
5224          \bool_gset_true:N \g_@@_empty_cell_bool
5225          \tl_gput_right:Ne \g_@@_HVdotsfor_lines_tl
5226            {
5227              \@@_Vdotsfor:nnnn
5228                { \int_use:N \c@iRow }
5229                { \int_use:N \c@jCol }
5230                { #2 }
5231                {
5232                  #1 , #3 ,
5233                  down = \exp_not:n { #4 } ,
5234                  up = \exp_not:n { #5 } ,
5235                  middle = \exp_not:n { #6 }
5236                }
5237            }
5238        }
5239    }
```

```
5240  \cs_new_protected:Npn \@@_Vdotsfor:nnnn #1 #2 #3 #4
5241    {
5242      \bool_set_false:N \l_@@_initial_open_bool
5243      \bool_set_false:N \l_@@_final_open_bool
```

For the column, it's easy.

```
5244      \int_set:Nn \l_@@_initial_j_int { #2 }
5245      \int_set_eq:NN \l_@@_final_j_int \l_@@_initial_j_int
```

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

```
5246      \int_compare:nNnTF { #1 } = \c_one_int
5247        {
5248          \int_set_eq:NN \l_@@_initial_i_int \c_one_int
5249          \bool_set_true:N \l_@@_initial_open_bool
5250        }
5251        {
5252          \cs_if_exist:cTF
5253            {
5254              pgf @ sh @ ns @ \@@_env:
5255              - \int_eval:n { #1 - 1 }
5256              - \int_use:N \l_@@_initial_j_int
5257            }
5258            { \int_set:Nn \l_@@_initial_i_int { #1 - 1 } }
5259            {
5260              \int_set:Nn \l_@@_initial_i_int { #1 }
5261              \bool_set_true:N \l_@@_initial_open_bool
5262            }
```

```
5263        }
5264      \int_compare:nNnTF { #1 + #3 -1 } = \c@iRow
5265        {
5266          \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5267          \bool_set_true:N \l_@@_final_open_bool
5268        }
5269        {
5270          \cs_if_exist:cTF
5271            {
5272              pgf @ sh @ ns @ \@@_env:
5273              - \int_eval:n { #1 + #3 }
5274              - \int_use:N \l_@@_final_j_int
5275            }
5276            { \int_set:Nn \l_@@_final_i_int { #1 + #3 } }
5277            {
5278              \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5279              \bool_set_true:N \l_@@_final_open_bool
5280            }
5281        }
5282      \group_begin:
5283      \@@_open_shorten:
5284      \int_if_zero:nTF { #2 }
5285        { \color { nicematrix-first-col } }
5286        {
5287          \int_compare:nNnT { #2 } = \g_@@_col_total_int
5288            { \color { nicematrix-last-col } }
5289        }
5290      \keys_set:nn { nicematrix / xdots } { #4 }
5291      \tl_if_empty:oF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
5292      \@@_actually_draw_Vdots:
5293      \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).

```
5294      \int_step_inline:nnn { #1 } { #1 + #3 - 1 }
5295        { \cs_set:cpn { @@ _ dotted _ ##1 - #2 } { } }
5296    }
```

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

```
5297  \NewDocumentCommand \@@_rotate: { O { } }
5298    {
5299      \peek_remove_spaces:n
5300        {
5301          \bool_gset_true:N \g_@@_rotate_bool
5302          \keys_set:nn { nicematrix / rotate } { #1 }
5303        }
5304    }
```

```
5305  \keys_define:nn { nicematrix / rotate }
5306    {
5307      c .code:n = \bool_gset_true:N \g_@@_rotate_c_bool ,
5308      c .value_forbidden:n = true ,
5309      unknown .code:n = \@@_error:n { Unknown~key~for~rotate }
5310    }
```

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, if also works with names of blocks.

First, we write a command with the following behaviour:

- If the argument is of the format i-j, our command applies the command \int_eval:n to i and j ;

- If not (that is to say, when it's a name of a \Block), the argument is left unchanged.

This must *not* be protected (and is, of course fully expandable).[13]

```
5311 \cs_new:Npn \@@_double_int_eval:n #1-#2 \q_stop
5312   {
5313     \tl_if_empty:nTF { #2 }
5314       { #1 }
5315       { \@@_double_int_eval_i:n #1-#2 \q_stop }
5316   }
5317 \cs_new:Npn \@@_double_int_eval_i:n #1-#2- \q_stop
5318   { \int_eval:n { #1 } - \int_eval:n { #2 } }
```

With the following construction, the command \@@_double_int_eval:n is applied to both arguments before the application of \@@_line_i:nn (the construction uses the fact the \@@_line_i:nn is protected and that \@@_double_int_eval:n is fully expandable).

```
5319 \hook_gput_code:nnn { begindocument } { . }
5320   {
5321     \cs_set_nopar:Npn \l_@@_argspec_tl
5322       { O { } m m ! O { } E { _ ^ : } { { } { } { } } }
5323     \tl_set_rescan:Nno  \l_@@_argspec_tl { } \l_@@_argspec_tl
5324     \exp_args:NNo \NewDocumentCommand \@@_line \l_@@_argspec_tl
5325       {
5326         \group_begin:
5327         \keys_set:nn { nicematrix / xdots } { #1 , #4 , down = #5 , up = #6 }
5328         \tl_if_empty:oF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
5329           \use:e
5330             {
5331               \@@_line_i:nn
5332                 { \@@_double_int_eval:n #2 - \q_stop }
5333                 { \@@_double_int_eval:n #3 - \q_stop }
5334             }
5335         \group_end:
5336       }
5337   }
5338 \cs_new_protected:Npn \@@_line_i:nn #1 #2
5339   {
5340     \bool_set_false:N \l_@@_initial_open_bool
5341     \bool_set_false:N \l_@@_final_open_bool
5342     \bool_lazy_or:nnTF
5343       { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #1 } }
5344       { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #2 } }
5345       { \@@_error:nnn { unknown~cell~for~line~in~CodeAfter } { #1 } { #2 } }
```

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

```
5346         { \legacy_if:nF { measuring@ } { \@@_draw_line_ii:nn { #1 } { #2 } } }
5347   }
```

---

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

```
5348  \hook_gput_code:nnn { begindocument } { . }
5349    {
5350      \cs_new_protected:Npx \@@_draw_line_ii:nn #1 #2
5351        {
```

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:.

```
5352          \c_@@_pgfortikzpicture_tl
5353          \@@_draw_line_iii:nn { #1 } { #2 }
5354          \c_@@_endpgfortikzpicture_tl
5355        }
5356    }
```

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

```
5357  \cs_new_protected:Npn \@@_draw_line_iii:nn #1 #2
5358    {
5359      \pgfrememberpicturepositiononpagetrue
5360      \pgfpointshapeborder { \@@_env: - #1 } { \@@_qpoint:n { #2 } }
5361      \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
5362      \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
5363      \pgfpointshapeborder { \@@_env: - #2 } { \@@_qpoint:n { #1 } }
5364      \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
5365      \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
5366      \@@_draw_line:
5367    }
```

The commands \Ldots, \Cdots, \Vdots, \Ddots, and \Iddots don't use this command because they
have to do other settings (for example, the diagonal lines must be parallelized).

21 The command \RowStyle

\g_@@_row_style_tl may contain several instructions of the form:

   \@@_if_row_less_than:nn { number } { instructions }

Then, \g_@@_row_style_tl will be inserted in all the cells of the array (and also in both components
of a \diagbox in a cell of in a mono-row block).
The test \@@_if_row_less_then:nn ensures that the instructions are inserted only if you are in a
row which is (still) in the scope of that instructions (which depends on the value of the key nb-rows
of \RowStyle).
That test will be active even in an expandable context because \@@_if_row_less_then:nn is *not*
protected.
#1 is the first row *after* the scope of the instructions in #2

```
5368  \cs_new:Npn \@@_if_row_less_than:nn #1 #2
5369    { \int_compare:nNnT { \c@iRow } < { #1 } { #2 } }
```

\@@_put_in_row_style will be used several times by \RowStyle.

```
5370  \cs_set_protected:Npn \@@_put_in_row_style:n #1
5371    {
5372      \tl_gput_right:Ne \g_@@_row_style_tl
5373        {
```

Be careful, \exp_not:N \@@_if_row_less_than:nn can't be replaced by a protected version of
\@@_if_row_less_than:nn.

```
5374          \exp_not:N
5375          \@@_if_row_less_than:nn
5376            { \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int } }
```

The `\scan_stop:` is mandatory (for ex. for the case where `\rotate` is used in the argument of `\RowStyle`).

```
5377              { \exp_not:n { #1 } \scan_stop: }
5378        }
5379    }
5380  \cs_generate_variant:Nn \@@_put_in_row_style:n { e }

5381  \keys_define:nn { nicematrix / RowStyle }
5382    {
5383      cell-space-top-limit .dim_set:N = \l_tmpa_dim ,
5384      cell-space-top-limit .value_required:n = true ,
5385      cell-space-bottom-limit .dim_set:N = \l_tmpb_dim ,
5386      cell-space-bottom-limit .value_required:n = true ,
5387      cell-space-limits .meta:n =
5388        {
5389          cell-space-top-limit = #1 ,
5390          cell-space-bottom-limit = #1 ,
5391        } ,
5392      color .tl_set:N = \l_@@_color_tl ,
5393      color .value_required:n = true ,
5394      bold .bool_set:N = \l_@@_bold_row_style_bool ,
5395      bold .default:n = true ,
5396      nb-rows .code:n =
5397        \str_if_eq:nnTF { #1 } { * }
5398          { \int_set:Nn \l_@@_key_nb_rows_int { 500 } }
5399          { \int_set:Nn \l_@@_key_nb_rows_int { #1 } } ,
5400      nb-rows .value_required:n = true ,
5401      rowcolor .tl_set:N = \l_tmpa_tl ,
5402      rowcolor .value_required:n = true ,
5403      unknown .code:n = \@@_error:n { Unknown~key~for~RowStyle }
5404    }


5405  \NewDocumentCommand \@@_RowStyle:n { O { } m }
5406    {
5407      \group_begin:
5408      \tl_clear:N \l_tmpa_tl
5409      \tl_clear:N \l_@@_color_tl
5410      \int_set_eq:NN \l_@@_key_nb_rows_int \c_one_int
5411      \dim_zero:N \l_tmpa_dim
5412      \dim_zero:N \l_tmpb_dim
5413      \keys_set:nn { nicematrix / RowStyle } { #1 }
```

If the key `rowcolor` has been used.

```
5414      \tl_if_empty:NF \l_tmpa_tl
5415        {
```

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

```
5416        \tl_gput_right:Ne \g_@@_pre_code_before_tl
5417          {
```

The command `\@@_exp_color_arg:No` is *fully expandable*.

```
5418          \@@_exp_color_arg:No \@@_rectanglecolor \l_tmpa_tl
5419            { \int_use:N \c@iRow - \int_use:N \c@jCol }
5420            { \int_use:N \c@iRow - * }
5421          }
```

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

```
5422        \int_compare:nNnT \l_@@_key_nb_rows_int > \c_one_int
5423          {
5424          \tl_gput_right:Ne \g_@@_pre_code_before_tl
5425            {
5426            \@@_exp_color_arg:No \@@_rowcolor \l_tmpa_tl
5427              {
```

```
5428                    \int_eval:n { \c@iRow + 1 }
5429                    - \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int - 1 }
5430                  }
5431              }
5432          }
5433        }
5434      \@@_put_in_row_style:n { \exp_not:n { #2 } }
```
\l_tmpa_dim is the value of the key cell-space-top-limit of \RowStyle.
```
5435      \dim_compare:nNnT \l_tmpa_dim > \c_zero_dim
5436        {
5437          \exp_args:Ne \@@_put_in_row_style:n
5438            {
5439              \tl_gput_right:Nn \exp_not:N \g_@@_cell_after_hook_tl
5440                {
```
It's not possible to chanage the following code by using \dim_set_eq:NN (because of expansion).
```
5441                  \dim_set:Nn \l_@@_cell_space_top_limit_dim
5442                    { \dim_use:N \l_tmpa_dim }
5443                }
5444            }
5445        }
```
\l_tmpb_dim is the value of the key cell-space-bottom-limit of \RowStyle.
```
5446      \dim_compare:nNnT \l_tmpb_dim > \c_zero_dim
5447        {
5448          \exp_args:Ne \@@_put_in_row_style:n
5449            {
5450              \tl_gput_right:Nn \exp_not:N \g_@@_cell_after_hook_tl
5451                {
5452                  \dim_set:Nn \l_@@_cell_space_bottom_limit_dim
5453                    { \dim_use:N \l_tmpb_dim }
5454                }
5455            }
5456        }
```
\l_@@_color_tl is the value of the key color of \RowStyle.
```
5457      \tl_if_empty:NF \l_@@_color_tl
5458        {
5459          \@@_put_in_row_style:e
5460            {
5461              \mode_leave_vertical:
5462              \@@_color:n { \l_@@_color_tl }
5463            }
5464        }
```
\l_@@_bold_row_style_bool is the value of the key bold.
```
5465      \bool_if:NT \l_@@_bold_row_style_bool
5466        {
5467          \@@_put_in_row_style:n
5468            {
5469              \exp_not:n
5470                {
5471                  \if_mode_math:
5472                    \c_math_toggle_token
5473                    \bfseries \boldmath
5474                    \c_math_toggle_token
5475                  \else:
5476                    \bfseries \boldmath
5477                  \fi:
5478                }
5479            }
5480        }
5481      \group_end:
5482      \g_@@_row_style_tl
5483      \ignorespaces
```

```
5484        }
```

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).

```
5485  \cs_new_protected:Npn \@@_add_to_colors_seq:nn #1 #2
5486    {
```

Firt, 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.

```
5487        \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.

```
5488        \str_if_in:nnF { #1 } { !! }
5489          {
5490            \seq_map_indexed_inline:Nn \g_@@_colors_seq
5491              { \tl_if_eq:nnT { #1 } { ##2 } { \int_set:Nn \l_tmpa_int { ##1 } } } }
5492          }
5493        \int_if_zero:nTF \l_tmpa_int
```

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

```
5494          {
5495            \seq_gput_right:Nn \g_@@_colors_seq { #1 }
5496            \tl_gset:cx { g_@@_color _ \seq_count:N \g_@@_colors_seq _ tl } { #2 }
5497          }
```

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

```
5498        { \tl_gput_right:cx { g_@@_color _ \int_use:N \l_tmpa_int _tl } { #2 } }
5499    }
```

```
5500  \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e n }
5501  \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e e }
```

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

```
5502  \cs_new_protected:Npn \@@_clip_with_rounded_corners:
5503    {
5504      \dim_compare:nNnT \l_@@_tab_rounded_corners_dim > \c_zero_dim
5505        {
```

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

```
5506            \group_begin:
5507            \pgfsetcornersarced
5508              {
5509                \pgfpoint
5510                  { \l_@@_tab_rounded_corners_dim }
5511                  { \l_@@_tab_rounded_corners_dim }
5512              }
```

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.

```
5513            \bool_if:NTF \l_@@_hvlines_bool
5514              {
5515                \pgfpathrectanglecorners
5516                  {
5517                    \pgfpointadd
5518                      { \@@_qpoint:n { row-1 } }
5519                      { \pgfpoint { 0.5 \arrayrulewidth } { \c_zero_dim } }
5520                  }
5521                  {
5522                    \pgfpointadd
5523                      {
5524                        \@@_qpoint:n
5525                          { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
5526                      }
5527                      { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
5528                  }
5529              }
5530              {
5531                \pgfpathrectanglecorners
5532                  { \@@_qpoint:n { row-1 } }
5533                  {
5534                    \pgfpointadd
5535                      {
5536                        \@@_qpoint:n
5537                          { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
5538                      }
5539                      { \pgfpoint \c_zero_dim \arrayrulewidth }
5540                  }
5541              }
5542            \pgfusepath { clip }
5543            \group_end:
```

The TeX group was for `\pgfsetcornersarced`.

```
5544          }
5545      }
```

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`).

```
5546  \cs_new_protected:Npn \@@_actually_color:
5547    {
5548        \pgfpicture
5549        \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.

```
5550        \@@_clip_with_rounded_corners:
5551        \seq_map_indexed_inline:Nn \g_@@_colors_seq
5552          {
5553            \int_compare:nNnTF { ##1 } = \c_one_int
```

```
5554              {
5555                \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_nocolor:n
5556                \use:c { g_@@_color _ 1 _tl }
5557                \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_normal:n
5558              }
5559              {
5560                \begin { pgfscope }
5561                  \@@_color_opacity ##2
5562                  \use:c { g_@@_color _ ##1 _tl }
5563                  \tl_gclear:c { g_@@_color _ ##1 _tl }
5564                  \pgfusepath { fill }
5565                \end { pgfscope }
5566              }
5567          }
5568      \endpgfpicture
5569    }
```

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

```
5570 \cs_new_protected:Npn \@@_color_opacity
5571    {
5572      \peek_meaning:NTF [
5573        { \@@_color_opacity:w }
5574        { \@@_color_opacity:w [ ] }
5575    }
```

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.

```
5576 \cs_new_protected:Npn \@@_color_opacity:w [ #1 ]
5577    {
5578      \tl_clear:N \l_tmpa_tl
5579      \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.

```
5580      \tl_if_empty:NF \l_tmpa_tl { \exp_args:No \pgfsetfillopacity \l_tmpa_tl }
5581      \tl_if_empty:NTF \l_tmpb_tl
5582        { \@declaredcolor }
5583        { \use:e { \exp_not:N \@undeclaredcolor [ \l_tmpb_tl ] } } }
5584    }
```

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

```
5585 \keys_define:nn { nicematrix / color-opacity }
5586    {
5587      opacity .tl_set:N          = \l_tmpa_tl ,
5588      opacity .value_required:n = true
5589    }
```

```
5590 \cs_new_protected:Npn \@@_cartesian_color:nn #1 #2
5591    {
5592      \cs_set_nopar:Npn \l_@@_rows_tl { #1 }
5593      \cs_set_nopar:Npn \l_@@_cols_tl { #2 }
5594      \@@_cartesian_path:
5595    }
```

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

```
5596 \NewDocumentCommand \@@_rowcolor { O { } m m }
5597    {
5598      \tl_if_blank:nF { #2 }
5599        {
```

```
5600        \@@_add_to_colors_seq:en
5601          { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5602          { \@@_cartesian_color:nn { #3 } { - } }
5603      }
5604    }
```

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

```
5605  \NewDocumentCommand \@@_columncolor { O { } m m }
5606    {
5607      \tl_if_blank:nF { #2 }
5608        {
5609          \@@_add_to_colors_seq:en
5610            { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5611            { \@@_cartesian_color:nn { - } { #3 } }
5612        }
5613    }
```

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

```
5614  \NewDocumentCommand \@@_rectanglecolor { O { } m m m }
5615    {
5616      \tl_if_blank:nF { #2 }
5617        {
5618          \@@_add_to_colors_seq:en
5619            { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5620            { \@@_rectanglecolor:nnn { #3 } { #4 } { \c_zero_dim } }
5621        }
5622    }
```

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

```
5623  \NewDocumentCommand \@@_roundedrectanglecolor { O { } m m m m }
5624    {
5625      \tl_if_blank:nF { #2 }
5626        {
5627          \@@_add_to_colors_seq:en
5628            { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5629            { \@@_rectanglecolor:nnn { #3 } { #4 } { #5 } }
5630        }
5631    }
```

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

```
5632  \cs_new_protected:Npn \@@_rectanglecolor:nnn #1 #2 #3
5633    {
5634      \@@_cut_on_hyphen:w #1 \q_stop
5635      \tl_clear_new:N \l_@@_tmpc_tl
5636      \tl_clear_new:N \l_@@_tmpd_tl
5637      \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
5638      \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
5639      \@@_cut_on_hyphen:w #2 \q_stop
5640      \tl_set:Ne \l_@@_rows_tl { \l_@@_tmpc_tl - \l_tmpa_tl }
5641      \tl_set:Ne \l_@@_cols_tl { \l_@@_tmpd_tl - \l_tmpb_tl }
```

The command `\@@_cartesian_path:n` takes in two implicit arguments: `\l_@@_cols_tl` and `\l_@@_rows_tl`.

```
5642      \@@_cartesian_path:n { #3 }
5643    }
```

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

```
5644  \NewDocumentCommand \@@_cellcolor { O { } m m }
5645    {
5646      \clist_map_inline:nn { #3 }
5647        { \@@_rectanglecolor [ #1 ] { #2 } { ##1 } { ##1 } }
5648    }
```

```
5649  \NewDocumentCommand \@@_chessboardcolors { O { } m m  }
5650    {
5651      \int_step_inline:nn \c@iRow
5652        {
5653          \int_step_inline:nn \c@jCol
5654            {
5655              \int_if_even:nTF { ####1 + ##1 }
5656                { \@@_cellcolor [ #1 ] { #2 } }
5657                { \@@_cellcolor [ #1 ] { #3 } }
5658              { ##1 - ####1 }
5659            }
5660        }
5661    }
```

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".

```
5662  \NewDocumentCommand \@@_arraycolor { O { } m }
5663    {
5664      \@@_rectanglecolor [ #1 ] { #2 }
5665        { 1 - 1 }
5666        { \int_use:N \c@iRow - \int_use:N \c@jCol }
5667    }
```

```
5668  \keys_define:nn { nicematrix / rowcolors }
5669    {
5670      respect-blocks .bool_set:N = \l_@@_respect_blocks_bool ,
5671      respect-blocks .default:n = true ,
5672      cols .tl_set:N = \l_@@_cols_tl ,
5673      restart .bool_set:N = \l_@@_rowcolors_restart_bool ,
5674      restart .default:n = true ,
5675      unknown .code:n = \@@_error:n { Unknown~key~for~rowcolors }
5676    }
```

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 commmand `\@@_rowcolors` appears as a special case of `\@@_rowlistcolors`.

#1 (optional) is the color space; #2 is a list of intervals of rows; #3 is the list of colors; #4 is for the optional list of pairs *key=value*.

```
5677  \NewDocumentCommand \@@_rowlistcolors { O { } m m O { } }
5678    {
```

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

```
5679      \group_begin:
5680      \seq_clear_new:N \l_@@_colors_seq
5681      \seq_set_split:Nnn \l_@@_colors_seq { , } { #3 }
5682      \tl_clear_new:N \l_@@_cols_tl
5683      \cs_set_nopar:Npn \l_@@_cols_tl { - }
5684      \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).

```
5685      \int_zero_new:N \l_@@_color_int
5686      \int_set_eq:NN \l_@@_color_int \c_one_int
5687      \bool_if:NT \l_@@_respect_blocks_bool
5688        {
```

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).

```
5689          \seq_set_eq:NN \l_tmpb_seq \g_@@_pos_of_blocks_seq
5690          \seq_set_filter:NNn \l_tmpa_seq \l_tmpb_seq
5691            { \@@_not_in_exterior_p:nnnnn ##1 }
5692        }
5693      \pgfpicture
5694      \pgf@relevantforpicturesizefalse
```

#2 is the list of intervals of rows.

```
5695      \clist_map_inline:nn { #2 }
5696        {
5697          \cs_set_nopar:Npn \l_tmpa_tl { ##1 }
5698          \tl_if_in:NnTF \l_tmpa_tl { - }
5699            { \@@_cut_on_hyphen:w ##1 \q_stop }
5700            { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }
```

Now, l_tmpa_tl and l_tmpb_tl are the first row and the last row of the interval of rows that we have to treat. The counter \l_tmpa_int will be the index of the loop over the rows.

```
5701          \int_set:Nn \l_tmpa_int \l_tmpa_tl
5702          \int_set:Nn \l_@@_color_int
5703            { \bool_if:NTF \l_@@_rowcolors_restart_bool 1 \l_tmpa_tl }
5704          \int_zero_new:N \l_@@_tmpc_int
5705          \int_set:Nn \l_@@_tmpc_int \l_tmpb_tl
5706          \int_do_until:nNnn \l_tmpa_int > \l_@@_tmpc_int
5707            {
```

We will compute in \l_tmpb_int the last row of the "block".

```
5708              \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).

```
5709              \bool_if:NT \l_@@_respect_blocks_bool
5710                {
5711                  \seq_set_filter:NNn \l_tmpb_seq \l_tmpa_seq
5712                    { \@@_intersect_our_row_p:nnnnn ####1 }
5713                  \seq_map_inline:Nn \l_tmpb_seq { \@@_rowcolors_i:nnnnn ####1 }
```

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

```
5714                }
5715              \tl_set:No \l_@@_rows_tl
5716                { \int_use:N \l_tmpa_int - \int_use:N \l_tmpb_int }
```

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

```
5717              \tl_clear_new:N \l_@@_color_tl
5718              \tl_set:Ne \l_@@_color_tl
5719                {
5720                  \@@_color_index:n
5721                    {
5722                      \int_mod:nn
5723                        { \l_@@_color_int - 1 }
5724                        { \seq_count:N \l_@@_colors_seq }
5725                      + 1
5726                    }
5727                }
5728              \tl_if_empty:NF \l_@@_color_tl
5729                {
5730                  \@@_add_to_colors_seq:ee
5731                    { \tl_if_blank:nF { #1 } { [ #1 ] } { \l_@@_color_tl } }
5732                    { \@@_cartesian_color:nn { \l_@@_rows_tl } { \l_@@_cols_tl } }
5733                }
5734              \int_incr:N \l_@@_color_int
5735              \int_set:Nn \l_tmpa_int { \l_tmpb_int + 1 }
5736            }
5737        }
5738      \endpgfpicture
```

```
5739        \group_end:
5740    }
```

The command `\@@_color_index:n` peeks in `\l_@@_colors_seq` the color at the index `#1`. However, if that color is the symbol `=`, the previous one is poken. This macro is recursive.

```
5741  \cs_new:Npn \@@_color_index:n #1
5742    {
5743      \str_if_eq:eeTF { \seq_item:Nn \l_@@_colors_seq { #1 } } { = }
5744        { \@@_color_index:n { #1 - 1 } }
5745        { \seq_item:Nn \l_@@_colors_seq { #1 } }
5746    }
```

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

```
5747  \NewDocumentCommand \@@_rowcolors { O { } m m m }
5748    { \@@_rowlistcolors [ #1 ] { #2 } { { #3 } , { #4 } } }
```

The braces around `#3` and `#4` are mandatory.

```
5749  \cs_new_protected:Npn \@@_rowcolors_i:nnnnn #1 #2 #3 #4 #5
5750    {
5751      \int_compare:nNnT { #3 } > \l_tmpb_int
5752        { \int_set:Nn \l_tmpb_int { #3 } }
5753    }
```

```
5754  \prg_new_conditional:Nnn \@@_not_in_exterior:nnnnn p
5755    {
5756      \int_if_zero:nTF { #4 }
5757        \prg_return_false:
5758        {
5759          \int_compare:nNnTF { #2 } > \c@jCol
5760            \prg_return_false:
5761            \prg_return_true:
5762        }
5763    }
```

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

```
5764  \prg_new_conditional:Nnn \@@_intersect_our_row:nnnnn p
5765    {
5766      \int_compare:nNnTF { #1 } > \l_tmpa_int
5767        \prg_return_false:
5768        {
5769          \int_compare:nNnTF \l_tmpa_int > { #3 }
5770            \prg_return_false:
5771            \prg_return_true:
5772        }
5773    }
```

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`).

```
5774  \cs_new_protected:Npn \@@_cartesian_path_normal:n #1
5775    {
5776      \dim_compare:nNnTF { #1 } = \c_zero_dim
5777        {
5778          \bool_if:NTF
```

```
5779          \l_@@_nocolor_used_bool
5780          \@@_cartesian_path_normal_ii:
5781          {
5782            \seq_if_empty:NTF \l_@@_corners_cells_seq
5783              { \@@_cartesian_path_normal_i:n { #1 } }
5784              \@@_cartesian_path_normal_ii:
5785          }
5786        }
5787        { \@@_cartesian_path_normal_i:n { #1 } }
5788    }
```

First, the situation where is a rectangular zone of cells will be colored as a whole (in the instructions of the resulting PDF). The argument is the radius of the corners.

```
5789  \cs_new_protected:Npn \@@_cartesian_path_normal_i:n #1
5790    {
5791      \pgfsetcornersarced { \pgfpoint { #1 } { #1 } }
```

We begin the loop over the columns.

```
5792      \clist_map_inline:Nn \l_@@_cols_tl
5793        {
5794          \cs_set_nopar:Npn \l_tmpa_tl { ##1 }
5795          \tl_if_in:NnTF \l_tmpa_tl { - }
5796            { \@@_cut_on_hyphen:w ##1 \q_stop }
5797            { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
5798          \tl_if_empty:NTF \l_tmpa_tl
5799            { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5800            {
5801              \tl_if_eq:NNT \l_tmpa_tl \c_@@_star_tl
5802                { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5803            }
5804          \tl_if_empty:NTF \l_tmpb_tl
5805            { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
5806            {
5807              \tl_if_eq:NNT \l_tmpb_tl \c_@@_star_tl
5808                { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
5809            }
5810          \int_compare:nNnT \l_tmpb_tl > \g_@@_col_total_int
5811            { \tl_set:No \l_tmpb_tl { \int_use:N \g_@@_col_total_int } }
```

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

```
5812          \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
5813          \@@_qpoint:n { col - \l_tmpa_tl }
5814          \int_compare:nNnTF \l_@@_first_col_int = \l_tmpa_tl
5815            { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
5816            { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
5817          \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 }  }
5818          \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }
```

We begin the loop over the rows.

```
5819          \clist_map_inline:Nn \l_@@_rows_tl
5820            {
5821              \cs_set_nopar:Npn \l_tmpa_tl { ####1 }
5822              \tl_if_in:NnTF \l_tmpa_tl { - }
5823                { \@@_cut_on_hyphen:w ####1 \q_stop }
5824                { \@@_cut_on_hyphen:w ####1 - ####1 \q_stop }
5825              \tl_if_empty:NTF \l_tmpa_tl
5826                { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5827                {
5828                  \tl_if_eq:NNT \l_tmpa_tl \c_@@_star_tl
5829                    { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5830                }
5831              \tl_if_empty:NTF \l_tmpb_tl
5832                { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }
5833                {
```

```
5834                \tl_if_eq:NNT \l_tmpb_tl \c_@@_star_tl
5835                  { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }
5836              }
5837            \int_compare:nNnT \l_tmpb_tl > \g_@@_row_total_int
5838              { \tl_set:No \l_tmpb_tl { \int_use:N \g_@@_row_total_int } } }
```

Now, the numbers of both rows are in `\l_tmpa_tl` and `\l_tmpb_tl`.

```
5839                \cs_if_exist:cF
5840                  { @@ _ \l_tmpa_tl _ \l_@@_tmpc_tl _ nocolor }
5841                  {
5842                    \@@_qpoint:n { row - \int_eval:n { \l_tmpb_tl + 1 } }
5843                    \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
5844                    \@@_qpoint:n { row - \l_tmpa_tl }
5845                    \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
5846                    \pgfpathrectanglecorners
5847                      { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
5848                      { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5849                  }
5850              }
5851          }
5852      }
```

Now, the case where the cells will be colored cell by cell (it's mandatory for example if the key `corners` is used).

```
5853  \cs_new_protected:Npn \@@_cartesian_path_normal_ii:
5854    {
5855      \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
5856      \@@_expand_clist:NN \l_@@_rows_tl \c@iRow
```

We begin the loop over the columns.

```
5857      \clist_map_inline:Nn \l_@@_cols_tl
5858        {
5859          \@@_qpoint:n { col - ##1 }
5860          \int_compare:nNnTF \l_@@_first_col_int = { ##1 }
5861            { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
5862            { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
5863          \@@_qpoint:n { col - \int_eval:n { ##1 + 1 }  }
5864          \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }
```

We begin the loop over the rows.

```
5865          \clist_map_inline:Nn \l_@@_rows_tl
5866            {
5867              \seq_if_in:NnF \l_@@_corners_cells_seq
5868                { ####1 - ##1 }
5869                {
5870                  \@@_qpoint:n { row - \int_eval:n { ####1 + 1 } }
5871                  \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
5872                  \@@_qpoint:n { row - ####1 }
5873                  \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
5874                  \cs_if_exist:cF { @@ _ ####1 _ ##1 _ nocolor }
5875                    {
5876                      \pgfpathrectanglecorners
5877                        { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
5878                        { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5879                    }
5880                }
5881            }
5882        }
5883    }
```

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`).

```
5884  \cs_new_protected:Npn \@@_cartesian_path: { \@@_cartesian_path:n \c_zero_dim }
```

Despite its name, the following command does not create a PGF path. It declares as colored by the "empty color" all the cells in what would be the path. Hence, the other coloring instructions of nicematrix won't put color in those cells. the

```
5885 \cs_new_protected:Npn \@@_cartesian_path_nocolor:n #1
5886   {
5887     \bool_set_true:N \l_@@_nocolor_used_bool
5888     \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
5889     \@@_expand_clist:NN \l_@@_rows_tl \c@iRow
```

We begin the loop over the columns.

```
5890     \clist_map_inline:Nn \l_@@_rows_tl
5891       {
5892         \clist_map_inline:Nn \l_@@_cols_tl
5893           { \cs_set:cpn { @@ _ ##1 _ ####1 _ nocolor } { } }
5894       }
5895   }
```

The following command will be used only with \l_@@_cols_tl and \c@jCol (first case) or with \l_@@_rows_tl and \c@iRow (second case). For instance, with \l_@@_cols_tl equal to 2,4-6,8-* and \c@jCol equal to 10, the clist \l_@@_cols_tl will be replaced by 2,4,5,6,8,9,10.

```
5896 \cs_new_protected:Npn \@@_expand_clist:NN #1 #2
5897   {
5898     \clist_set_eq:NN \l_tmpa_clist #1
5899     \clist_clear:N #1
5900     \clist_map_inline:Nn \l_tmpa_clist
5901       {
5902         \cs_set_nopar:Npn \l_tmpa_tl { ##1 }
5903         \tl_if_in:NnTF \l_tmpa_tl { - }
5904           { \@@_cut_on_hyphen:w ##1 \q_stop }
5905           { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
5906         \bool_lazy_or:nnT
5907           { \tl_if_blank_p:o \l_tmpa_tl }
5908           { \str_if_eq_p:on \l_tmpa_tl { * } }
5909           { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5910         \bool_lazy_or:nnT
5911           { \tl_if_blank_p:o \l_tmpb_tl }
5912           { \str_if_eq_p:on \l_tmpb_tl { * } }
5913           { \tl_set:No \l_tmpb_tl { \int_use:N #2 } }
5914         \int_compare:nNnT \l_tmpb_tl > #2
5915           { \tl_set:No \l_tmpb_tl { \int_use:N #2 } }
5916         \int_step_inline:nnn \l_tmpa_tl \l_tmpb_tl
5917           { \clist_put_right:Nn #1 { ####1 } }
5918       }
5919   }
```

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

```
5920 \NewDocumentCommand \@@_cellcolor_tabular { O { } m }
5921   {
5922     \@@_test_color_inside:
5923     \tl_gput_right:Ne \g_@@_pre_code_before_tl
5924       {
```

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).

```
5925         \@@_cellcolor [ #1 ] { \exp_not:n { #2 } }
5926           { \int_use:N \c@iRow - \int_use:N \c@jCol }
5927       }
5928     \ignorespaces
5929   }
```

139

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

```
5930  \NewDocumentCommand \@@_rowcolor_tabular { O { } m }
5931    {
5932      \@@_test_color_inside:
5933      \tl_gput_right:Ne \g_@@_pre_code_before_tl
5934        {
5935          \@@_rectanglecolor [ #1 ] { \exp_not:n { #2 } }
5936            { \int_use:N \c@iRow - \int_use:N \c@jCol }
5937            { \int_use:N \c@iRow - \exp_not:n { \int_use:N \c@jCol } }
5938        }
5939      \ignorespaces
5940    }
```

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).

```
5941  \NewDocumentCommand { \@@_rowcolors_tabular } { O { } m m }
5942    { \@@_rowlistcolors_tabular [ #1 ] { { #2 } , { #3 } } }
```

The braces around `#2` and `#3` are mandatory.

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

```
5943  \NewDocumentCommand { \@@_rowlistcolors_tabular } { O { } m O { } }
5944    {
5945      \@@_test_color_inside:
5946      \peek_remove_spaces:n
5947        { \@@_rowlistcolors_tabular:nnn { #1 } { #2 } { #3 } }
5948    }
```

```
5949  \cs_new_protected:Npn \@@_rowlistcolors_tabular:nnn #1 #2 #3
5950    {
```

A use of `\rowlistcolors` in the tabular erases the instructions `\rowlistcolors` which are in force. However, it's possible to put *several* instructions `\rowlistcolors` in the same row of a tabular: it may be useful when those instructions `\rowlistcolors` concerns different columns of the tabular (thanks to the key `cols` of `\rowlistcolors`). That's why we store the different instructions `\rowlistcolors` which are in force in a sequence `\g_@@_rowlistcolors_seq`. Now, we will filter that sequence to keep only the elements which have been issued on the actual row. We will store the elements to keep in the `\g_tmpa_seq`.

```
5951      \seq_gclear:N \g_tmpa_seq
5952      \seq_map_inline:Nn \g_@@_rowlistcolors_seq
5953        { \@@_rowlistcolors_tabular_i:nnnn ##1 }
5954      \seq_gset_eq:NN \g_@@_rowlistcolors_seq \g_tmpa_seq
```

Now, we add to the sequence `\g_@@_rowlistcolors_seq` (which is the list of the commands `\rowlistcolors` which are in force) the current instruction `\rowlistcolors`.

```
5955      \seq_gput_right:Ne \g_@@_rowlistcolors_seq
5956        {
5957          { \int_use:N \c@iRow }
5958          { \exp_not:n { #1 } }
5959          { \exp_not:n { #2 } }
5960          { restart , cols = \int_use:N \c@jCol - , \exp_not:n { #3 } }
5961        }
5962    }
```

The following command will be applied to each component of `\g_@@_rowlistcolors_seq`. Each component of that sequence is a kind of 4-uple of the form `{#1}{#2}{#3}{#4}`.
`#1` is the number of the row where the command `\rowlistcolors` has been issued.
`#2` is the colorimetric space (optional argument of the `\rowlistcolors`).
`#3` is the list of colors (mandatory argument of `\rowlistcolors`).

#4 is the list of *key=value* pairs (last optional argument of \rowlistcolors).

```
5963 \cs_new_protected:Npn \@@_rowlistcolors_tabular_i:nnnn #1 #2 #3 #4
5964   {
5965     \int_compare:nNnTF { #1 } = \c@iRow
```

We (temporary) keep in memory in \g_tmpa_seq the instructions which will still be in force after the current instruction (because they have been issued in the same row of the tabular).

```
5966       { \seq_gput_right:Nn \g_tmpa_seq { { #1 } { #2 } { #3 } { #4 } } }
5967       {
5968         \tl_gput_right:Ne \g_@@_pre_code_before_tl
5969           {
5970             \@@_rowlistcolors
5971               [ \exp_not:n { #2 } ]
5972               { #1 - \int_eval:n { \c@iRow - 1 } }
5973               { \exp_not:n { #3 } }
5974               [ \exp_not:n { #4 } ]
5975           }
5976       }
5977   }
```

The following command will be used at the end of the tabular, just before the execution of the \g_@@_pre_code_before_tl. It clears the sequence \g_@@_rowlistcolors_seq of all the commands \rowlistcolors which are (still) in force.

```
5978 \cs_new_protected:Npn \@@_clear_rowlistcolors_seq:
5979   {
5980     \seq_map_inline:Nn \g_@@_rowlistcolors_seq
5981       { \@@_rowlistcolors_tabular_ii:nnnn ##1 }
5982     \seq_gclear:N \g_@@_rowlistcolors_seq
5983   }
```

```
5984 \cs_new_protected:Npn \@@_rowlistcolors_tabular_ii:nnnn #1 #2 #3 #4
5985   {
5986     \tl_gput_right:Nn \g_@@_pre_code_before_tl
5987       { \@@_rowlistcolors [ #2 ] { #1 } { #3 } [ #4 ] }
5988   }
```

The first mandatory argument of the command \@@_rowlistcolors which is writtent in the pre-\CodeBefore is of the form i: it means that the command must be applied to all the rows from the row *i* until the end of the tabular.

```
5989 \NewDocumentCommand \@@_columncolor_preamble { O { } m }
5990   {
```

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).

```
5991     \int_compare:nNnT \c@jCol > \g_@@_col_total_int
5992       {
```

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).

```
5993         \tl_gput_left:Ne \g_@@_pre_code_before_tl
5994           {
5995             \exp_not:N \columncolor [ #1 ]
5996               { \exp_not:n { #2 } } { \int_use:N \c@jCol }
5997           }
5998       }
5999   }
```

```
6000 \hook_gput_code:nnn { begindocument } { . }
6001   {
6002     \IfPackageLoadedTF { colortbl }
6003       {
6004         \cs_set_eq:NN \@@_old_cellcolor \cellcolor
6005         \cs_set_eq:NN \@@_old_rowcolor \rowcolor
6006         \cs_new_protected:Npn \@@_revert_colortbl:
6007           {
6008             \hook_gput_code:nnn { env / tabular / begin } { nicematrix }
6009               {
6010                 \cs_set_eq:NN \cellcolor \@@_old_cellcolor
6011                 \cs_set_eq:NN \rowcolor \@@_old_rowcolor
6012               }
6013           }
6014       }
6015       { \cs_new_protected:Npn \@@_revert_colortbl: { } }
6016   }
```

23 The vertical and horizontal rules

OnlyMainNiceMatrix

We give to the user the possibility to define new types of columns (with `\newcolumntype` 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`.

```
6017 \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`.

```
6018 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix:n #1
6019   {
6020     \int_if_zero:nTF \l_@@_first_col_int
6021       { \@@_OnlyMainNiceMatrix_i:n { #1 } }
6022       {
6023         \int_if_zero:nTF \c@jCol
6024           {
6025             \int_compare:nNnF \c@iRow = { -1 }
6026               { \int_compare:nNnF \c@iRow = { \l_@@_last_row_int - 1 } { #1 } }
6027           }
6028           { \@@_OnlyMainNiceMatrix_i:n { #1 } }
6029       }
6030   }
```

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.

```
6031 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix_i:n #1
6032   {
6033     \int_if_zero:nF \c@iRow
6034       {
6035         \int_compare:nNnF \c@iRow = \l_@@_last_row_int
6036           {
```

```
6037            \int_compare:nNnT \c@jCol > \c_zero_int
6038              { \bool_if:NF \l_@@_in_last_col_bool { #1 } }
6039          }
6040        }
6041    }
```

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.

```
6042 \keys_define:nn { nicematrix / Rules }
6043    {
6044      position .int_set:N = \l_@@_position_int ,
6045      position .value_required:n = true ,
6046      start .int_set:N = \l_@@_start_int ,
6047      end .code:n =
6048        \bool_lazy_or:nnTF
6049          { \tl_if_empty_p:n { #1 } }
6050          { \str_if_eq_p:nn { #1 } { last } }
6051          { \int_set_eq:NN \l_@@_end_int \c@jCol }
6052          { \int_set:Nn \l_@@_end_int { #1 } } }
6053    }
```

It's possible that the rule won't be drawn continuously from **start** ot **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.

```
6054 \keys_define:nn { nicematrix / RulesBis }
6055    {
6056      multiplicity .int_set:N = \l_@@_multiplicity_int ,
6057      multiplicity .initial:n = 1 ,
6058      dotted .bool_set:N = \l_@@_dotted_bool ,
6059      dotted .initial:n = false ,
6060      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].

```
6061      color .code:n =
6062        \@@_set_CT@arc@:n { #1 }
6063        \tl_set:Nn \l_@@_rule_color_tl { #1 } ,
6064      color .value_required:n = true ,
6065      sep-color .code:n = \@@_set_CT@drsc@:n { #1 } ,
6066      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.

```
6067      tikz .code:n =
6068        \IfPackageLoadedTF { tikz }
6069          { \clist_put_right:Nn \l_@@_tikz_rule_tl { #1 } }
6070          { \@@_error:n { tikz~without~tikz } } ,
6071      tikz .value_required:n = true ,
6072      total-width .dim_set:N = \l_@@_rule_width_dim ,
```

```
6073        total-width .value_required:n = true ,
6074        width .meta:n = { total-width = #1 } ,
6075        unknown .code:n = \@@_error:n { Unknow~key~for~RulesBis }
6076      }
```

The vertical rules

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

```
6077  \cs_new_protected:Npn \@@_vline:n #1
6078    {
```

The group is for the options.

```
6079        \group_begin:
6080        \int_set_eq:NN \l_@@_end_int \c@iRow
6081        \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).

```
6082        \int_compare:nNnT \l_@@_position_int < { \c@jCol + 2 }
6083          \@@_vline_i:
6084        \group_end:
6085      }
```

```
6086  \cs_new_protected:Npn \@@_vline_i:
6087    {
```

`\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`.

```
6088        \tl_set:No \l_tmpb_tl { \int_use:N \l_@@_position_int }
6089        \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
6090          \l_tmpa_tl
6091          {
```

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.

```
6092            \bool_gset_true:N \g_tmpa_bool
6093            \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6094              { \@@_test_vline_in_block:nnnnn ##1 }
6095            \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
6096              { \@@_test_vline_in_block:nnnnn ##1 }
6097            \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
6098              { \@@_test_vline_in_stroken_block:nnnn ##1 }
6099            \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_v:
6100            \bool_if:NTF \g_tmpa_bool
6101              {
6102                \int_if_zero:nT \l_@@_local_start_int
```

We keep in memory that we have a rule to draw. `\l_@@_local_start_int` will be the starting row of the rule that we will have to draw.

```
6103                  { \int_set:Nn \l_@@_local_start_int \l_tmpa_tl }
6104              }
6105              {
6106                \int_compare:nNnT \l_@@_local_start_int > \c_zero_int
6107                  {
6108                    \int_set:Nn \l_@@_local_end_int { \l_tmpa_tl - 1 }
6109                    \@@_vline_ii:
6110                    \int_zero:N \l_@@_local_start_int
6111                  }
6112              }
6113          }
6114        \int_compare:nNnT \l_@@_local_start_int > \c_zero_int
6115          {
```

144

```
6116        \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
6117        \@@_vline_ii:
6118      }
6119  }


6120  \cs_new_protected:Npn \@@_test_in_corner_v:
6121    {
6122      \int_compare:nNnTF \l_tmpb_tl = { \int_eval:n { \c@jCol + 1 } }
6123        {
6124          \seq_if_in:NeT
6125            \l_@@_corners_cells_seq
6126            { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
6127            { \bool_set_false:N \g_tmpa_bool }
6128        }
6129        {
6130          \seq_if_in:NeT
6131            \l_@@_corners_cells_seq
6132            { \l_tmpa_tl - \l_tmpb_tl }
6133            {
6134              \int_compare:nNnTF \l_tmpb_tl = \c_one_int
6135                { \bool_set_false:N \g_tmpa_bool }
6136                {
6137                  \seq_if_in:NeT
6138                    \l_@@_corners_cells_seq
6139                    { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
6140                    { \bool_set_false:N \g_tmpa_bool }
6141                }
6142            }
6143        }
6144    }


6145  \cs_new_protected:Npn \@@_vline_ii:
6146    {
6147      \tl_clear:N \l_@@_tikz_rule_tl
6148      \keys_set:nV { nicematrix / RulesBis } \l_@@_other_keys_tl
6149      \bool_if:NTF \l_@@_dotted_bool
6150        \@@_vline_iv:
6151        {
6152          \tl_if_empty:NTF \l_@@_tikz_rule_tl
6153            \@@_vline_iii:
6154            \@@_vline_v:
6155        }
6156    }
```

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

```
6157  \cs_new_protected:Npn \@@_vline_iii:
6158    {
6159      \pgfpicture
6160      \pgfrememberpicturepositiononpagetrue
6161      \pgf@relevantforpicturesizefalse
6162      \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6163      \dim_set_eq:NN \l_tmpa_dim \pgf@y
6164      \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6165      \dim_set:Nn \l_tmpb_dim
6166        {
6167          \pgf@x
6168          - 0.5 \l_@@_rule_width_dim
6169          +
6170          ( \arrayrulewidth * \l_@@_multiplicity_int
6171            + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6172        }
```

```
6173    \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6174    \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
6175    \bool_lazy_all:nT
6176      {
6177        { \int_compare_p:nNn \l_@@_multiplicity_int > \c_one_int }
6178        { \cs_if_exist_p:N \CT@drsc@ }
6179        { ! \tl_if_blank_p:o \CT@drsc@ }
6180      }
6181      {
6182        \group_begin:
6183        \CT@drsc@
6184        \dim_add:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }
6185        \dim_sub:Nn \l_@@_tmpc_dim { 0.5 \arrayrulewidth }
6186        \dim_set:Nn \l_@@_tmpd_dim
6187          {
6188            \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
6189            * ( \l_@@_multiplicity_int - 1 )
6190          }
6191        \pgfpathrectanglecorners
6192          { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6193          { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
6194        \pgfusepath { fill }
6195        \group_end:
6196      }
6197    \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6198    \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
6199    \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6200      {
6201        \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
6202        \dim_sub:Nn \l_tmpb_dim \doublerulesep
6203        \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6204        \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
6205      }
6206    \CT@arc@
6207    \pgfsetlinewidth { 1.1 \arrayrulewidth }
6208    \pgfsetrectcap
6209    \pgfusepathqstroke
6210    \endpgfpicture
6211  }
```

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

```
6212 \cs_new_protected:Npn \@@_vline_iv:
6213  {
6214    \pgfpicture
6215    \pgfrememberpicturepositiononpagetrue
6216    \pgf@relevantforpicturesizefalse
6217    \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6218    \dim_set:Nn \l_@@_x_initial_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
6219    \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
6220    \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6221    \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
6222    \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6223    \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
6224    \CT@arc@
6225    \@@_draw_line:
6226    \endpgfpicture
6227  }
```

The following code is for the case when the user uses the key tikz.

```
6228 \cs_new_protected:Npn \@@_vline_v:
6229  {
6230    \begin {tikzpicture }
```

By default, the color defined by `\arrayrulecolor` or by `rules/color` will be used, but it's still possible to change the color by using the key `color` or, of course, the key `color` inside the key `tikz` (that is to say the key `color` provided by PGF.

```
6231     \CT@arc@
6232     \tl_if_empty:NF \l_@@_rule_color_tl
6233       { \tl_put_right:Ne \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6234     \pgfrememberpicturepositiononpagetrue
6235     \pgf@relevantforpicturesizefalse
6236     \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6237     \dim_set_eq:NN \l_tmpa_dim \pgf@y
6238     \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6239     \dim_set:Nn \l_tmpb_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
6240     \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6241     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
6242     \exp_args:No \tikzset \l_@@_tikz_rule_tl
6243     \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6244       ( \l_tmpb_dim , \l_tmpa_dim ) --
6245       ( \l_tmpb_dim , \l_@@_tmpc_dim ) ;
6246     \end { tikzpicture }
6247   }
```

The command `\@@_draw_vlines:` draws all the vertical rules excepted in the blocks, in the virtual blocks (determined by a command such as `\Cdots`) and in the corners (if the key `corners` is used).

```
6248 \cs_new_protected:Npn \@@_draw_vlines:
6249   {
6250     \int_step_inline:nnn
6251       { \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool 2 1 }
6252       {
6253         \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool
6254           \c@jCol
6255           { \int_eval:n { \c@jCol + 1 } }
6256       }
6257       {
6258         \tl_if_eq:NNF \l_@@_vlines_clist \c_@@_all_tl
6259           { \clist_if_in:NnT \l_@@_vlines_clist { ##1 } }
6260           { \@@_vline:n { position = ##1 , total-width = \arrayrulewidth } }
6261       }
6262   }
```

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}.

```
6263 \cs_new_protected:Npn \@@_hline:n #1
6264   {
```

The group is for the options.

```
6265     \group_begin:
6266     \int_zero_new:N \l_@@_end_int
6267     \int_set_eq:NN \l_@@_end_int \c@jCol
6268     \keys_set_known:nnN { nicematrix / Rules } { #1 } \l_@@_other_keys_tl
6269     \@@_hline_i:
6270     \group_end:
6271   }
6272 \cs_new_protected:Npn \@@_hline_i:
6273   {
6274     \int_zero_new:N \l_@@_local_start_int
6275     \int_zero_new:N \l_@@_local_end_int
```

$\backslash$l_tmpa_tl is the number of row and $\backslash$l_tmpb_tl the number of column. When we have found a column corresponding to a rule to draw, we note its number in $\backslash$l_@@_tmpc_tl.

```
6276        \tl_set:No \l_tmpa_tl { \int_use:N \l_@@_position_int }
6277        \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
6278          \l_tmpb_tl
6279            {
```

The boolean $\backslash$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 $\backslash$Block or a virtual block corresponding to a dotted line, created by $\backslash$Cdots, $\backslash$Vdots, etc.), we will set $\backslash$g_tmpa_bool to false and the small horizontal rule won't be drawn.

```
6280              \bool_gset_true:N \g_tmpa_bool
6281              \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6282                { \@@_test_hline_in_block:nnnnn ##1 }
6283              \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
6284                { \@@_test_hline_in_block:nnnnn ##1 }
6285              \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
6286                { \@@_test_hline_in_stroken_block:nnnn ##1 }
6287              \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_h:
6288              \bool_if:NTF \g_tmpa_bool
6289                {
6290                  \int_if_zero:nT \l_@@_local_start_int
```

We keep in memory that we have a rule to draw. $\backslash$l_@@_local_start_int will be the starting row of the rule that we will have to draw.

```
6291                    { \int_set:Nn \l_@@_local_start_int \l_tmpb_tl }
6292                }
6293                {
6294                  \int_compare:nNnT \l_@@_local_start_int > \c_zero_int
6295                    {
6296                      \int_set:Nn \l_@@_local_end_int { \l_tmpb_tl - 1 }
6297                      \@@_hline_ii:
6298                      \int_zero:N \l_@@_local_start_int
6299                    }
6300                }
6301            }
6302        \int_compare:nNnT \l_@@_local_start_int > \c_zero_int
6303          {
6304            \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
6305            \@@_hline_ii:
6306          }
6307    }


6308  \cs_new_protected:Npn \@@_test_in_corner_h:
6309    {
6310      \int_compare:nNnTF \l_tmpa_tl = { \int_eval:n { \c@iRow + 1 } }
6311        {
6312          \seq_if_in:NeT
6313            \l_@@_corners_cells_seq
6314            { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
6315            { \bool_set_false:N \g_tmpa_bool }
6316        }
6317        {
6318          \seq_if_in:NeT
6319            \l_@@_corners_cells_seq
6320            { \l_tmpa_tl - \l_tmpb_tl }
6321            {
6322              \int_compare:nNnTF \l_tmpa_tl = \c_one_int
6323                { \bool_set_false:N \g_tmpa_bool }
6324                {
6325                  \seq_if_in:NeT
6326                    \l_@@_corners_cells_seq
6327                    { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
```

```
6328                 { \bool_set_false:N \g_tmpa_bool }
6329               }
6330             }
6331         }
6332     }


6333 \cs_new_protected:Npn \@@_hline_ii:
6334   {
6335     \tl_clear:N \l_@@_tikz_rule_tl
6336     \keys_set:nV { nicematrix / RulesBis } \l_@@_other_keys_tl
6337     \bool_if:NTF \l_@@_dotted_bool
6338       \@@_hline_iv:
6339       {
6340         \tl_if_empty:NTF \l_@@_tikz_rule_tl
6341           \@@_hline_iii:
6342           \@@_hline_v:
6343       }
6344   }
```

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

```
6345 \cs_new_protected:Npn \@@_hline_iii:
6346   {
6347     \pgfpicture
6348     \pgfrememberpicturepositiononpagetrue
6349     \pgf@relevantforpicturesizefalse
6350     \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6351     \dim_set_eq:NN \l_tmpa_dim \pgf@x
6352     \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6353     \dim_set:Nn \l_tmpb_dim
6354       {
6355         \pgf@y
6356         - 0.5 \l_@@_rule_width_dim
6357         +
6358         ( \arrayrulewidth * \l_@@_multiplicity_int
6359           + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6360       }
6361     \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6362     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
6363     \bool_lazy_all:nT
6364       {
6365         { \int_compare_p:nNn \l_@@_multiplicity_int > \c_one_int }
6366         { \cs_if_exist_p:N \CT@drsc@ }
6367         { ! \tl_if_blank_p:o \CT@drsc@ }
6368       }
6369       {
6370         \group_begin:
6371         \CT@drsc@
6372         \dim_set:Nn \l_@@_tmpd_dim
6373           {
6374             \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
6375             * ( \l_@@_multiplicity_int - 1 )
6376           }
6377         \pgfpathrectanglecorners
6378           { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6379           { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
6380         \pgfusepathqfill
6381         \group_end:
6382       }
6383     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6384     \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
6385     \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6386       {
```

```
6387          \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
6388          \dim_sub:Nn \l_tmpb_dim \doublerulesep
6389          \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6390          \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
6391        }
6392      \CT@arc@
6393      \pgfsetlinewidth { 1.1 \arrayrulewidth }
6394      \pgfsetrectcap
6395      \pgfusepathqstroke
6396      \endpgfpicture
6397    }
```

The following code is for the case of a dotted rule (with our system of rounded dots). The aim is that, by standard the dotted line fits between square brackets (\hline doesn't).

```
\begin{bNiceMatrix}
1 & 2 & 3 & 4 \\
\hline
1 & 2 & 3 & 4 \\
\hdottedline
1 & 2 & 3 & 4
\end{bNiceMatrix}
```

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ \hline 1 & 2 & 3 & 4 \\ \hdashline 1 & 2 & 3 & 4 \end{bmatrix}$$

But, if the user uses `margin`, the dotted line extends to have the same width as a \hline.

```
\begin{bNiceMatrix}[margin]
1 & 2 & 3 & 4 \\
\hline
1 & 2 & 3 & 4 \\
\hdottedline
1 & 2 & 3 & 4
\end{bNiceMatrix}
```

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ \hline 1 & 2 & 3 & 4 \\ \hdashline 1 & 2 & 3 & 4 \end{bmatrix}$$

```
6398 \cs_new_protected:Npn \@@_hline_iv:
6399   {
6400     \pgfpicture
6401     \pgfrememberpicturepositiononpagetrue
6402     \pgf@relevantforpicturesizefalse
6403     \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6404     \dim_set:Nn \l_@@_y_initial_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
6405     \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
6406     \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6407     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
6408     \int_compare:nNnT \l_@@_local_start_int = \c_one_int
6409       {
6410         \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
6411         \bool_if:NF \g_@@_delims_bool
6412           { \dim_sub:Nn \l_@@_x_initial_dim \arraycolsep }
```

For reasons purely aesthetic, we do an adjustment in the case of a rounded bracket. The correction by `0.5 \l_@@_xdots_inter_dim` is *ad hoc* for a better result.

```
6413         \tl_if_eq:NnF \g_@@_left_delim_tl (
6414           { \dim_add:Nn \l_@@_x_initial_dim  { 0.5 \l_@@_xdots_inter_dim } }
6415       }
6416     \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6417     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
6418     \int_compare:nNnT \l_@@_local_end_int = \c@jCol
6419       {
6420         \dim_add:Nn \l_@@_x_final_dim \l_@@_right_margin_dim
6421         \bool_if:NF \g_@@_delims_bool
6422           { \dim_add:Nn \l_@@_x_final_dim \arraycolsep }
6423         \tl_if_eq:NnF \g_@@_right_delim_tl )
6424           { \dim_gsub:Nn \l_@@_x_final_dim { 0.5 \l_@@_xdots_inter_dim } }
6425       }
6426     \CT@arc@
6427     \@@_draw_line:
```

```
6428      \endpgfpicture
6429    }
```

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`).

```
6430 \cs_new_protected:Npn \@@_hline_v:
6431    {
6432      \begin { tikzpicture }
```

By default, the color defined by `\arrayrulecolor` or by `rules/color` will be used, but it's still possible to change the color by using the key `color` or, of course, the key `color` inside the key `tikz` (that is to say the key `color` provided by PGF.

```
6433      \CT@arc@
6434      \tl_if_empty:NF \l_@@_rule_color_tl
6435        { \tl_put_right:Ne \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6436      \pgfrememberpicturepositiononpagetrue
6437      \pgf@relevantforpicturesizefalse
6438      \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6439      \dim_set_eq:NN \l_tmpa_dim \pgf@x
6440      \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6441      \dim_set:Nn \l_tmpb_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
6442      \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6443      \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
6444      \exp_args:No \tikzset \l_@@_tikz_rule_tl
6445      \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6446        ( \l_tmpa_dim , \l_tmpb_dim ) --
6447        ( \l_@@_tmpc_dim , \l_tmpb_dim ) ;
6448      \end { tikzpicture }
6449    }
```

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).

```
6450 \cs_new_protected:Npn \@@_draw_hlines:
6451    {
6452      \int_step_inline:nnn
6453        { \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool 2 1 }
6454        {
6455          \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool
6456            \c@iRow
6457            { \int_eval:n { \c@iRow + 1 } }
6458        }
6459        {
6460          \tl_if_eq:NNF \l_@@_hlines_clist \c_@@_all_tl
6461            { \clist_if_in:NnT \l_@@_hlines_clist { ##1 } }
6462            { \@@_hline:n { position = ##1 , total-width = \arrayrulewidth } }
6463        }
6464    }
```

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

```
6465 \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.

```
6466 \cs_set:Npn \@@_Hline_i:n #1
6467    {
6468      \peek_remove_spaces:n
6469        {
6470          \peek_meaning:NTF \Hline
6471            { \@@_Hline_ii:nn { #1 + 1 } }
6472            { \@@_Hline_iii:n { #1 } }
6473        }
6474    }
```

```
6475 \cs_set:Npn \@@_Hline_ii:nn #1 #2 { \@@_Hline_i:n { #1 } }
6476 \cs_set:Npn \@@_Hline_iii:n #1
6477   { \@@_collect_options:n { \@@_Hline_iv:nn { #1 } } }
6478 \cs_set:Npn \@@_Hline_iv:nn #1 #2
6479   {
6480     \@@_compute_rule_width:n { multiplicity = #1 , #2 }
6481     \skip_vertical:N \l_@@_rule_width_dim
6482     \tl_gput_right:Ne \g_@@_pre_code_after_tl
6483       {
6484         \@@_hline:n
6485           {
6486             multiplicity = #1 ,
6487             position = \int_eval:n { \c@iRow + 1 } ,
6488             total-width = \dim_use:N \l_@@_rule_width_dim ,
6489             #2
6490           }
6491       }
6492     \egroup
6493   }
```

Customized rules defined by the final user

The final user can define a customized rule by using the key `custom-line` in `\NiceMatrixOptions`. That key takes in as value a list of *key=value* pairs.

The following command will create the customized rule (it is executed when the final user uses the key `custom-line`, for example in `\NiceMatrixOptions`).

```
6494 \cs_new_protected:Npn \@@_custom_line:n #1
6495   {
6496     \str_clear_new:N \l_@@_command_str
6497     \str_clear_new:N \l_@@_ccommand_str
6498     \str_clear_new:N \l_@@_letter_str
6499     \tl_clear_new:N \l_@@_other_keys_tl
6500     \keys_set_known:nnN { 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.

```
6501     \bool_lazy_all:nTF
6502       {
6503         { \str_if_empty_p:N \l_@@_letter_str }
6504         { \str_if_empty_p:N \l_@@_command_str }
6505         { \str_if_empty_p:N \l_@@_ccommand_str }
6506       }
6507       { \@@_error:n { No~letter~and~no~command } }
6508       { \exp_args:No \@@_custom_line_i:n \l_@@_other_keys_tl }
6509   }
6510 \keys_define:nn { nicematrix / custom-line }
6511   {
6512     letter .str_set:N = \l_@@_letter_str ,
6513     letter .value_required:n = true ,
6514     command .str_set:N = \l_@@_command_str ,
6515     command .value_required:n = true ,
6516     ccommand .str_set:N = \l_@@_ccommand_str ,
6517     ccommand .value_required:n = true ,
6518   }


6519 \cs_new_protected:Npn \@@_custom_line_i:n #1
6520   {
```

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

```
6521      \bool_set_false:N \l_@@_tikz_rule_bool
6522      \bool_set_false:N \l_@@_dotted_rule_bool
6523      \bool_set_false:N \l_@@_color_bool

6524      \keys_set:nn { nicematrix / custom-line-bis } { #1 }
6525      \bool_if:NT \l_@@_tikz_rule_bool
6526        {
6527          \IfPackageLoadedTF { tikz }
6528            { }
6529            { \@@_error:n { tikz~in~custom-line~without~tikz } }
6530          \bool_if:NT \l_@@_color_bool
6531            { \@@_error:n { color~in~custom-line~with~tikz } }
6532        }
6533      \bool_if:NT \l_@@_dotted_rule_bool
6534        {
6535          \int_compare:nNnT \l_@@_multiplicity_int > \c_one_int
6536            { \@@_error:n { key~multiplicity~with~dotted } }
6537        }
6538      \str_if_empty:NF \l_@@_letter_str
6539        {
6540          \int_compare:nTF { \str_count:N \l_@@_letter_str != 1 }
6541            { \@@_error:n { Several~letters } }
6542            {
6543              \exp_args:NnV \tl_if_in:NnTF
6544                \c_@@_forbidden_letters_str \l_@@_letter_str
6545                { \@@_error:ne { Forbidden~letter } \l_@@_letter_str }
6546                {
```

During the analyse of the preamble provided by the final user, our automaton, for the letter corresponding at the custom line, will directly use the following command that you define in the main hash table of TeX.

```
6547                  \cs_set:cpn { @@ _ \l_@@_letter_str } ##1
6548                    { \@@_v_custom_line:n { #1 } }
6549                }
6550            }
6551        }
6552      \str_if_empty:NF \l_@@_command_str { \@@_h_custom_line:n { #1 } }
6553      \str_if_empty:NF \l_@@_ccommand_str { \@@_c_custom_line:n { #1 } }
6554    }

6555 \tl_const:Nn \c_@@_forbidden_letters_tl { lcrpmbVX|()[]!@<> }
6556 \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.

```
6557 \keys_define:nn { nicematrix / custom-line-bis }
6558    {
6559      multiplicity .int_set:N = \l_@@_multiplicity_int ,
6560      multiplicity .initial:n = 1 ,
6561      multiplicity .value_required:n = true ,
6562      color .code:n = \bool_set_true:N \l_@@_color_bool ,
6563      color .value_required:n = true ,
6564      tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6565      tikz .value_required:n = true ,
6566      dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6567      dotted .value_forbidden:n = true ,
6568      total-width .code:n = { } ,
6569      total-width .value_required:n = true ,
6570      width .code:n = { } ,
6571      width .value_required:n = true ,
```

153

```
6572        sep-color .code:n = { } ,
6573        sep-color .value_required:n = true ,
6574        unknown .code:n = \@@_error:n { Unknown~key~for~custom-line }
6575    }
```

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

```
6576  \bool_new:N \l_@@_dotted_rule_bool
6577  \bool_new:N \l_@@_tikz_rule_bool
6578  \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`.

```
6579  \keys_define:nn { nicematrix / custom-line-width }
6580    {
6581      multiplicity .int_set:N = \l_@@_multiplicity_int ,
6582      multiplicity .initial:n = 1 ,
6583      multiplicity .value_required:n = true ,
6584      tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6585      total-width .code:n = \dim_set:Nn \l_@@_rule_width_dim { #1 }
6586                            \bool_set_true:N \l_@@_total_width_bool ,
6587      total-width .value_required:n = true ,
6588      width .meta:n = { total-width = #1 } ,
6589      dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6590    }
```

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`).

```
6591  \cs_new_protected:Npn \@@_h_custom_line:n #1
6592    {
```

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`).

```
6593      \cs_set:cpn { nicematrix - \l_@@_command_str } { \Hline [ #1 ] }
6594      \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_command_str
6595    }
```

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`).

```
6596  \cs_new_protected:Npn \@@_c_custom_line:n #1
6597    {
```

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

```
6598      \exp_args:Nc \NewExpandableDocumentCommand
6599        { nicematrix - \l_@@_ccommand_str }
6600        { O { } m }
6601        {
6602          \noalign
6603            {
6604              \@@_compute_rule_width:n { #1 , ##1 }
6605              \skip_vertical:n { \l_@@_rule_width_dim }
6606              \clist_map_inline:nn
6607                { ##2 }
6608                { \@@_c_custom_line_i:nn { #1 , ##1 } { ####1 } }
6609            }
6610        }
6611      \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_ccommand_str
6612    }
```

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*.

```
6613 \cs_new_protected:Npn \@@_c_custom_line_i:nn #1 #2
6614   {
6615     \str_if_in:nnTF { #2 } { - }
6616       { \@@_cut_on_hyphen:w #2 \q_stop }
6617       { \@@_cut_on_hyphen:w #2 - #2 \q_stop }
6618     \tl_gput_right:Ne \g_@@_pre_code_after_tl
6619       {
6620         \@@_hline:n
6621           {
6622             #1 ,
6623             start = \l_tmpa_tl ,
6624             end = \l_tmpb_tl ,
6625             position = \int_eval:n { \c@iRow + 1 } ,
6626             total-width = \dim_use:N \l_@@_rule_width_dim
6627           }
6628       }
6629   }
6630 \cs_new_protected:Npn \@@_compute_rule_width:n #1
6631   {
6632     \bool_set_false:N \l_@@_tikz_rule_bool
6633     \bool_set_false:N \l_@@_total_width_bool
6634     \bool_set_false:N \l_@@_dotted_rule_bool
6635     \keys_set_known:nn { nicematrix / custom-line-width } { #1 }
6636     \bool_if:NF \l_@@_total_width_bool
6637       {
6638         \bool_if:NTF \l_@@_dotted_rule_bool
6639           { \dim_set:Nn \l_@@_rule_width_dim { 2 \l_@@_xdots_radius_dim } }
6640           {
6641             \bool_if:NF \l_@@_tikz_rule_bool
6642               {
6643                 \dim_set:Nn \l_@@_rule_width_dim
6644                   {
6645                     \arrayrulewidth * \l_@@_multiplicity_int
6646                     + \doublerulesep * ( \l_@@_multiplicity_int - 1 )
6647                   }
6648               }
6649           }
6650       }
6651   }
6652 \cs_new_protected:Npn \@@_v_custom_line:n #1
6653   {
6654     \@@_compute_rule_width:n { #1 }
```

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

```
6655     \tl_gput_right:Ne \g_@@_array_preamble_tl
6656       { \exp_not:N ! { \skip_horizontal:n { \dim_use:N \l_@@_rule_width_dim } } }
6657     \tl_gput_right:Ne \g_@@_pre_code_after_tl
6658       {
6659         \@@_vline:n
6660           {
6661             #1 ,
6662             position = \int_eval:n { \c@jCol + 1 } ,
6663             total-width = \dim_use:N \l_@@_rule_width_dim
6664           }
6665       }
6666     \@@_rec_preamble:n
6667   }
6668 \@@_custom_line:n
6669   { 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`.

```
6670 \cs_new_protected:Npn \@@_test_hline_in_block:nnnnn #1 #2 #3 #4 #5
6671   {
6672     \int_compare:nNnT \l_tmpa_tl > { #1 }
6673       {
6674         \int_compare:nNnT \l_tmpa_tl < { #3 + 1 }
6675           {
6676             \int_compare:nNnT \l_tmpb_tl > { #2 - 1 }
6677               {
6678                 \int_compare:nNnT \l_tmpb_tl < { #4 + 1 }
6679                   { \bool_gset_false:N \g_tmpa_bool }
6680               }
6681           }
6682       }
6683   }
```

The same for vertical rules.

```
6684 \cs_new_protected:Npn \@@_test_vline_in_block:nnnnn #1 #2 #3 #4 #5
6685   {
6686     \int_compare:nNnT \l_tmpa_tl > { #1 - 1 }
6687       {
6688         \int_compare:nNnT \l_tmpa_tl < { #3 + 1 }
6689           {
6690             \int_compare:nNnT \l_tmpb_tl > { #2 }
6691               {
6692                 \int_compare:nNnT \l_tmpb_tl < { #4 + 1 }
6693                   { \bool_gset_false:N \g_tmpa_bool }
6694               }
6695           }
6696       }
6697   }
6698 \cs_new_protected:Npn \@@_test_hline_in_stroken_block:nnnn #1 #2 #3 #4
6699   {
6700     \int_compare:nNnT \l_tmpb_tl > { #2 - 1 }
6701       {
6702         \int_compare:nNnT \l_tmpb_tl < { #4 + 1 }
6703           {
6704             \int_compare:nNnTF \l_tmpa_tl = { #1 }
6705               { \bool_gset_false:N \g_tmpa_bool }
6706               {
6707                 \int_compare:nNnT \l_tmpa_tl = { #3 + 1 }
6708                   { \bool_gset_false:N \g_tmpa_bool }
6709               }
6710           }
6711       }
6712   }
6713 \cs_new_protected:Npn \@@_test_vline_in_stroken_block:nnnn #1 #2 #3 #4
6714   {
6715     \int_compare:nNnT \l_tmpa_tl > { #1 - 1 }
6716       {
6717         \int_compare:nNnT \l_tmpa_tl < { #3 + 1 }
6718           {
6719             \int_compare:nNnTF \l_tmpb_tl = { #2 }
6720               { \bool_gset_false:N \g_tmpa_bool }
6721               {
6722                 \int_compare:nNnT \l_tmpb_tl = { #4 + 1 }
6723                   { \bool_gset_false:N \g_tmpa_bool }
6724               }
```

156

```
6725                }
6726            }
6727    }
```

24 The empty corners

When the key `corners` is raised, the rules are not drawn in the corners; they are not colored and `\TikzEveryCell` does not apply. Of course, we have to compute the corners before we begin to draw the rules.

```
6728 \cs_new_protected:Npn \@@_compute_corners:
6729    {
```

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.

```
6730        \seq_clear_new:N \l_@@_corners_cells_seq
6731        \clist_map_inline:Nn \l_@@_corners_clist
6732            {
6733                \str_case:nnF { ##1 }
6734                    {
6735                        { NW }
6736                        { \@@_compute_a_corner:nnnnnn 1 1 1 1 \c@iRow \c@jCol }
6737                        { NE }
6738                        { \@@_compute_a_corner:nnnnnn 1 \c@jCol 1 { -1 } \c@iRow 1 }
6739                        { SW }
6740                        { \@@_compute_a_corner:nnnnnn \c@iRow 1 { -1 } 1 1 \c@jCol }
6741                        { SE }
6742                        { \@@_compute_a_corner:nnnnnn \c@iRow \c@jCol { -1 } { -1 } 1 1 }
6743                    }
6744                { \@@_error:nn { bad~corner } { ##1 } }
6745            }
```

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

```
6746        \seq_if_empty:NF \l_@@_corners_cells_seq
6747            {
```

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.

```
6748            \tl_gput_right:Ne \g_@@_aux_tl
6749                {
6750                    \seq_set_from_clist:Nn \exp_not:N \l_@@_corners_cells_seq
6751                        { \seq_use:Nnnn \l_@@_corners_cells_seq , , , }
6752                }
6753            }
6754    }
```

"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.

```
6755 \cs_new_protected:Npn \@@_compute_a_corner:nnnnnn #1 #2 #3 #4 #5 #6
6756    {
```

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.

```
6757        \bool_set_false:N \l_tmpa_bool
6758        \int_zero_new:N \l_@@_last_empty_row_int
6759        \int_set:Nn \l_@@_last_empty_row_int { #1 }
6760        \int_step_inline:nnnn { #1 } { #3 } { #5 }
6761          {
6762            \@@_test_if_cell_in_a_block:nn { ##1 } { \int_eval:n { #2 } }
6763            \bool_lazy_or:nnTF
6764              {
6765                \cs_if_exist_p:c
6766                  { pgf @ sh @ ns @ \@@_env: - ##1 - \int_eval:n { #2 } }
6767              }
6768              \l_tmpb_bool
6769              { \bool_set_true:N \l_tmpa_bool }
6770              {
6771                \bool_if:NF \l_tmpa_bool
6772                  { \int_set:Nn \l_@@_last_empty_row_int { ##1 } }
6773              }
6774          }
```

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

```
6775        \bool_set_false:N \l_tmpa_bool
6776        \int_zero_new:N \l_@@_last_empty_column_int
6777        \int_set:Nn \l_@@_last_empty_column_int { #2 }
6778        \int_step_inline:nnnn { #2 } { #4 } { #6 }
6779          {
6780            \@@_test_if_cell_in_a_block:nn { \int_eval:n { #1 } } { ##1 }
6781            \bool_lazy_or:nnTF
6782              \l_tmpb_bool
6783              {
6784                \cs_if_exist_p:c
6785                  { pgf @ sh @ ns @ \@@_env: - \int_eval:n { #1 } - ##1 }
6786              }
6787              { \bool_set_true:N \l_tmpa_bool }
6788              {
6789                \bool_if:NF \l_tmpa_bool
6790                  { \int_set:Nn \l_@@_last_empty_column_int { ##1 } }
6791              }
6792          }
```

Now, we loop over the rows.

```
6793        \int_step_inline:nnnn { #1 } { #3 } \l_@@_last_empty_row_int
6794          {
```

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

```
6795          \bool_set_false:N \l_tmpa_bool
6796          \int_step_inline:nnnn { #2 } { #4 } \l_@@_last_empty_column_int
6797            {
6798              \@@_test_if_cell_in_a_block:nn { ##1 } { ####1 }
6799              \bool_lazy_or:nnTF
6800                \l_tmpb_bool
6801                {
6802                  \cs_if_exist_p:c
6803                    { pgf @ sh @ ns @ \@@_env: - ##1 - ####1 }
6804                }
6805                { \bool_set_true:N \l_tmpa_bool }
6806                {
6807                  \bool_if:NF \l_tmpa_bool
6808                    {
6809                      \int_set:Nn \l_@@_last_empty_column_int { ####1 }
```

```
6810                      \seq_put_right:Nn
6811                        \l_@@_corners_cells_seq
6812                        { ##1 - ####1 }
6813                    }
6814                }
6815            }
6816        }
6817    }
```

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).

```
6818  \cs_new_protected:Npn \@@_test_if_cell_in_a_block:nn #1 #2
6819    {
6820      \int_set:Nn \l_tmpa_int { #1 }
6821      \int_set:Nn \l_tmpb_int { #2 }
6822      \bool_set_false:N \l_tmpb_bool
6823      \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6824        { \@@_test_if_cell_in_block:nnnnnnn \l_tmpa_int \l_tmpb_int ##1 }
6825    }
6826  \cs_set_protected:Npn \@@_test_if_cell_in_block:nnnnnnn #1 #2 #3 #4 #5 #6 #7
6827    {
6828      \int_compare:nNnF { #3 } > { #1 }
6829        {
6830          \int_compare:nNnF { #1 } > { #5 }
6831            {
6832              \int_compare:nNnF { #4 } > { #2 }
6833                {
6834                  \int_compare:nNnF { #2 } > { #6 }
6835                    { \bool_set_true:N \l_tmpb_bool }
6836                }
6837            }
6838        }
6839    }
```

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.

```
6840  \bool_new:N \l_@@_block_auto_columns_width_bool
```

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

```
6841  \keys_define:nn { nicematrix / NiceMatrixBlock }
6842    {
6843      auto-columns-width .code:n =
6844        {
6845          \bool_set_true:N \l_@@_block_auto_columns_width_bool
6846          \dim_gzero_new:N \g_@@_max_cell_width_dim
6847          \bool_set_true:N \l_@@_auto_columns_width_bool
6848        }
6849    }
```

```
6850  \NewDocumentEnvironment { NiceMatrixBlock } { ! O { } }
6851    {
6852      \int_gincr:N \g_@@_NiceMatrixBlock_int
6853      \dim_zero:N \l_@@_columns_width_dim
```

```
6854        \keys_set:nn { nicematrix / NiceMatrixBlock } { #1 }
6855        \bool_if:NT \l_@@_block_auto_columns_width_bool
6856          {
6857            \cs_if_exist:cT
6858              { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
6859              {
6860               % is \exp_args:NNe mandatory?
6861                \exp_args:NNe \dim_set:Nn \l_@@_columns_width_dim
6862                  {
6863                    \use:c
6864                      { @@_max_cell_width _ \int_use:N \g_@@_NiceMatrixBlock_int }
6865                  }
6866              }
6867          }
6868      }
```

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`).

```
6869      {
6870        \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.

```
6871          { \int_gdecr:N \g_@@_NiceMatrixBlock_int }
6872          {
6873            \bool_if:NT \l_@@_block_auto_columns_width_bool
6874              {
6875                \iow_shipout:Nn \@mainaux \ExplSyntaxOn
6876                \iow_shipout:Ne \@mainaux
6877                  {
6878                    \cs_gset:cpn
6879                      { @@ _ max _ cell _ width _ \int_use:N \g_@@_NiceMatrixBlock_int }
```

For technical reasons, we have to include the width of a potential rule on the right side of the cells.

```
6880                      { \dim_eval:n { \g_@@_max_cell_width_dim + \arrayrulewidth } }
6881                  }
6882                \iow_shipout:Nn \@mainaux \ExplSyntaxOff
6883              }
6884          }
6885      \ignorespacesafterend
6886    }
```

26 The extra nodes

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

```
6887 \cs_generate_variant:Nn \dim_min:nn { v n }
6888 \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).

```
6889 \cs_new_protected:Npn \@@_create_extra_nodes:
6890   {
6891     \bool_if:nTF \l_@@_medium_nodes_bool
6892       {
6893         \bool_if:NTF \l_@@_large_nodes_bool
6894           \@@_create_medium_and_large_nodes:
6895           \@@_create_medium_nodes:
6896       }
```

```
6897        { \bool_if:NT \l_@@_large_nodes_bool \@@_create_large_nodes: }
6898    }
```

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`.

```
6899  \cs_new_protected:Npn \@@_computations_for_medium_nodes:
6900    {
6901      \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6902        {
6903          \dim_zero_new:c { l_@@_row_\@@_i: _min_dim }
6904          \dim_set_eq:cN { l_@@_row_\@@_i: _min_dim } \c_max_dim
6905          \dim_zero_new:c { l_@@_row_\@@_i: _max_dim }
6906          \dim_set:cn { l_@@_row_\@@_i: _max_dim } { - \c_max_dim }
6907        }
6908      \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
6909        {
6910          \dim_zero_new:c { l_@@_column_\@@_j: _min_dim }
6911          \dim_set_eq:cN { l_@@_column_\@@_j: _min_dim } \c_max_dim
6912          \dim_zero_new:c { l_@@_column_\@@_j: _max_dim }
6913          \dim_set:cn { l_@@_column_\@@_j: _max_dim } { - \c_max_dim }
6914        }
```
We begin the two nested loops over the rows and the columns of the array.
```
6915      \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6916        {
6917          \int_step_variable:nnNn
6918            \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.
```
6919            {
6920              \cs_if_exist:cT
6921                { 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`.
```
6922                {
6923                  \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { south~west }
6924                  \dim_set:cn { l_@@_row_\@@_i: _min_dim}
6925                    { \dim_min:vn { l_@@_row _ \@@_i: _min_dim } \pgf@y }
6926                  \seq_if_in:NeF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
6927                    {
6928                      \dim_set:cn { l_@@_column _ \@@_j: _min_dim}
6929                        { \dim_min:vn { l_@@_column _ \@@_j: _min_dim } \pgf@x }
6930                    }
```

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`.

```
6931                    \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { north~east }
6932                    \dim_set:cn { l_@@_row _ \@@_i: _ max_dim }
6933                      { \dim_max:vn { l_@@_row _ \@@_i: _ max_dim } \pgf@y }
6934                    \seq_if_in:NeF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
6935                      {
6936                        \dim_set:cn { l_@@_column _ \@@_j: _ max_dim }
6937                          { \dim_max:vn { l_@@_column _ \@@_j: _max_dim } \pgf@x }
6938                      }
6939                  }
6940              }
6941          }
```

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

```
6942      \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6943        {
6944          \dim_compare:nNnT
6945            { \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } } = \c_max_dim
6946            {
6947              \@@_qpoint:n { row - \@@_i: - base }
6948              \dim_set:cn { l_@@_row _ \@@_i: _ max _ dim } \pgf@y
6949              \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim } \pgf@y
6950            }
6951        }
6952      \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
6953        {
6954          \dim_compare:nNnT
6955            { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } } = \c_max_dim
6956            {
6957              \@@_qpoint:n { col - \@@_j: }
6958              \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim } \pgf@y
6959              \dim_set:cn { l_@@_column _ \@@_j: _ min _ dim } \pgf@y
6960            }
6961        }
6962    }
```

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

```
6963 \cs_new_protected:Npn \@@_create_medium_nodes:
6964    {
6965      \pgfpicture
6966        \pgfrememberpicturepositiononpagetrue
6967        \pgf@relevantforpicturesizefalse
6968        \@@_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".

```
6969        \cs_set_nopar:Npn \l_@@_suffix_tl { -medium }
6970        \@@_create_nodes:
6971      \endpgfpicture
6972    }
```

The command `\@@_create_large_nodes:` must be used when we want to create only the "large nodes" and not the medium ones[14]. 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:`.

---

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

```
6973  \cs_new_protected:Npn \@@_create_large_nodes:
6974    {
6975      \pgfpicture
6976        \pgfrememberpicturepositiononpagetrue
6977        \pgf@relevantforpicturesizefalse
6978        \@@_computations_for_medium_nodes:
6979        \@@_computations_for_large_nodes:
6980        \cs_set_nopar:Npn \l_@@_suffix_tl { - large }
6981        \@@_create_nodes:
6982      \endpgfpicture
6983    }
6984  \cs_new_protected:Npn \@@_create_medium_and_large_nodes:
6985    {
6986      \pgfpicture
6987        \pgfrememberpicturepositiononpagetrue
6988        \pgf@relevantforpicturesizefalse
6989        \@@_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".

```
6990        \cs_set_nopar:Npn \l_@@_suffix_tl { - medium }
6991        \@@_create_nodes:
6992        \@@_computations_for_large_nodes:
6993        \cs_set_nopar:Npn \l_@@_suffix_tl { - large }
6994        \@@_create_nodes:
6995      \endpgfpicture
6996    }
```

For "large nodes", the exterior rows and columns don't interfer. 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.

```
6997  \cs_new_protected:Npn \@@_computations_for_large_nodes:
6998    {
6999      \int_set_eq:NN \l_@@_first_row_int \c_one_int
7000      \int_set_eq:NN \l_@@_first_col_int \c_one_int
```

We have to change the values of all the dimensions l_@@_row_i_min_dim, l_@@_row_i_max_dim, l_@@_column_j_min_dim and l_@@_column_j_max_dim.

```
7001      \int_step_variable:nNn { \c@iRow - 1 } \@@_i:
7002        {
7003          \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim }
7004            {
7005              (
7006                \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } +
7007                \dim_use:c { l_@@_row _ \int_eval:n { \@@_i: + 1 }  _ max _ dim }
7008              )
7009              / 2
7010            }
7011          \dim_set_eq:cc { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
7012            { l_@@_row_\@@_i: _min_dim }
7013        }
7014      \int_step_variable:nNn { \c@jCol - 1 } \@@_j:
7015        {
7016          \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim }
7017            {
7018              (
7019                \dim_use:c { l_@@_column _ \@@_j: _ max _ dim } +
7020                \dim_use:c
7021                  { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
7022              )
7023              / 2
7024            }
7025          \dim_set_eq:cc { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
7026            { l_@@_column _ \@@_j: _ max _ dim }
7027        }
```

163

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

```
7028        \dim_sub:cn
7029          { l_@@_column _ 1 _ min _ dim }
7030          \l_@@_left_margin_dim
7031        \dim_add:cn
7032          { l_@@_column _ \int_use:N \c@jCol _ max _ dim }
7033          \l_@@_right_margin_dim
7034      }
```

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`).

```
7035  \cs_new_protected:Npn \@@_create_nodes:
7036    {
7037      \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
7038        {
7039          \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
7040            {
```

We draw the rectangular node for the cell (`\@@_i:`-`\@@_j:`).

```
7041              \@@_pgf_rect_node:nnnnn
7042                { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7043                { \dim_use:c { l_@@_column_ \@@_j: _min_dim } }
7044                { \dim_use:c { l_@@_row_ \@@_i: _min_dim } }
7045                { \dim_use:c { l_@@_column_ \@@_j: _max_dim } }
7046                { \dim_use:c { l_@@_row_ \@@_i: _max_dim } }
7047              \str_if_empty:NF \l_@@_name_str
7048                {
7049                  \pgfnodealias
7050                    { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
7051                    { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7052                }
7053            }
7054        }
```

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 correspondant values of n.

```
7055        \seq_map_pairwise_function:NNN
7056          \g_@@_multicolumn_cells_seq
7057          \g_@@_multicolumn_sizes_seq
7058          \@@_node_for_multicolumn:nn
7059    }
```

```
7060  \cs_new_protected:Npn \@@_extract_coords_values: #1 - #2 \q_stop
7061    {
7062      \cs_set_nopar:Npn \@@_i: { #1 }
7063      \cs_set_nopar:Npn \@@_j: { #2 }
7064    }
```

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").

```
7065  \cs_new_protected:Npn \@@_node_for_multicolumn:nn #1 #2
7066    {
7067      \@@_extract_coords_values: #1 \q_stop
7068      \@@_pgf_rect_node:nnnnn
7069        { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7070        { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } }
```

```
7071        { \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } }
7072        { \dim_use:c { l_@@_column _ \int_eval:n { \@@_j: +#2-1 } _ max _ dim } }
7073        { \dim_use:c { l_@@_row _ \@@_i: _ max _ dim } }
7074      \str_if_empty:NF \l_@@_name_str
7075        {
7076          \pgfnodealias
7077            { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
7078            { \int_use:N \g_@@_env_int - \@@_i: - \@@_j: \l_@@_suffix_tl}
7079        }
7080    }
```

27 The blocks

The following code deals with the command `\Block`. This command has no direct link with the environment `{NiceMatrixBlock}`.

The options of the command `\Block` will be analyzed first in the cell of the array (and once again when the block will be put in the array). Here is the set of keys for the first pass (in the cell of the array).

```
7081  \keys_define:nn { nicematrix / Block / FirstPass }
7082    {
7083      j .code:n = \str_set:Nn \l_@@_hpos_block_str j
7084                 \bool_set_true:N \l_@@_p_block_bool ,
7085      j .value_forbidden:n = true ,
7086      l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7087      l .value_forbidden:n = true ,
7088      r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7089      r .value_forbidden:n = true ,
7090      c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7091      c .value_forbidden:n = true ,
7092      L .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7093      L .value_forbidden:n = true ,
7094      R .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7095      R .value_forbidden:n = true ,
7096      C .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7097      C .value_forbidden:n = true ,
7098      t .code:n = \str_set:Nn \l_@@_vpos_block_str t ,
7099      t .value_forbidden:n = true ,
7100      T .code:n = \str_set:Nn \l_@@_vpos_block_str T ,
7101      T .value_forbidden:n = true ,
7102      b .code:n = \str_set:Nn \l_@@_vpos_block_str b ,
7103      b .value_forbidden:n = true ,
7104      B .code:n = \str_set:Nn \l_@@_vpos_block_str B ,
7105      B .value_forbidden:n = true ,
7106      m .code:n = \str_set:Nn \l_@@_vpos_block_str c ,
7107      m .value_forbidden:n = true ,
7108      v-center .meta:n = m ,
7109      p .code:n = \bool_set_true:N \l_@@_p_block_bool ,
7110      p .value_forbidden:n = true ,
7111      color .code:n =
7112        \@@_color:n { #1 }
7113        \tl_set_rescan:Nnn
7114          \l_@@_draw_tl
7115          { \char_set_catcode_other:N ! }
7116          { #1 } ,
7117      color .value_required:n = true ,
7118      respect-arraystretch .code:n =
7119        \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
7120      respect-arraystretch .value_forbidden:n = true ,
```

```
7121    }
```

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.

```
7122  \cs_new_protected:Npn \@@_Block: { \@@_collect_options:n { \@@_Block_i: } }
```

```
7123  \NewExpandableDocumentCommand \@@_Block_i: { m m D < > { } +m }
7124    {
```

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).

```
7125      \peek_remove_spaces:n
7126        {
7127          \tl_if_blank:nTF { #2 }
7128            { \@@_Block_ii:nnnnn \c_one_int \c_one_int }
7129            {
7130              \int_compare:nNnTF { \char_value_catcode:n { 45 } } = { 13 }
7131              \@@_Block_i_czech \@@_Block_i
7132              #2 \q_stop
7133            }
7134          { #1 } { #3 } { #4 }
7135        }
7136    }
```

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

```
7137  \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`.

```
7138  {
7139    \char_set_catcode_active:N -
7140    \cs_new:Npn \@@_Block_i_czech #1-#2 \q_stop { \@@_Block_ii:nnnnn { #1 } { #2 } }
7141  }
```

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.

```
7142  \cs_new_protected:Npn \@@_Block_ii:nnnnn #1 #2 #3 #4 #5
7143    {
```

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).

```
7144      \bool_lazy_or:nnTF
7145        { \tl_if_blank_p:n { #1 } }
7146        { \str_if_eq_p:Vn \c_@@_star_str { #1 } }
7147        { \int_set:Nn \l_tmpa_int { 100 } }
7148        { \int_set:Nn \l_tmpa_int { #1 } }
7149      \bool_lazy_or:nnTF
7150        { \tl_if_blank_p:n { #2 } }
7151        { \str_if_eq_p:Vn \c_@@_star_str { #2 } }
7152        { \int_set:Nn \l_tmpb_int { 100 } }
7153        { \int_set:Nn \l_tmpb_int { #2 } }
```

If the block is mono-column.

```
7154      \int_compare:nNnTF \l_tmpb_int = \c_one_int
7155        {
7156          \tl_if_empty:NTF \l_@@_hpos_cell_tl
7157            { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_c_str }
7158            { \str_set:NV \l_@@_hpos_block_str \l_@@_hpos_cell_tl }
7159        }
7160        { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_c_str }
```

The value of `\l_@@_hpos_block_str` may be modified by the keys of the command `\Block` that we will analyze now.

```
7161      \keys_set_known:nn { nicematrix / Block / FirstPass } { #3 }

7162    \tl_set:Ne \l_tmpa_tl
7163      {
7164        { \int_use:N \c@iRow }
7165        { \int_use:N \c@jCol }
7166        { \int_eval:n { \c@iRow + \l_tmpa_int - 1 } }
7167        { \int_eval:n { \c@jCol + \l_tmpb_int - 1 } }
7168      }
```

Now, `\l_tmpa_tl` contains an "object" corresponding to the position of the block with four components, each of them surrounded by curly brackets:
{*imin*}{*jmin*}{*imax*}{*jmax*}.

We have different treatments when the key p is used and when the block is mono-column or mono-row, etc. That's why we have several macros: `\@@_Block_iv:nnnnn`, `\@@_Block_v:nnnnn`, `\@@_Block_vi:nnnn`, etc. (the five arguments of those macros are provided by curryfication).

```
7169      \bool_set_false:N \l_tmpa_bool
7170      \bool_if:NT \l_@@_amp_in_blocks_bool
7171        { \tl_if_in:nnT { #5 } { & } { \bool_set_true:N \l_tmpa_bool } }
7172      \bool_case:nF
7173        {
7174          \l_tmpa_bool                { \exp_args:Nee \@@_Block_vii:nnnnn }
7175          \l_@@_p_block_bool          { \exp_args:Nee \@@_Block_vi:nnnnn }
```

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.

```
7176          \l_@@_X_bool                { \exp_args:Nee \@@_Block_v:nnnnn }
7177          { \tl_if_empty_p:n { #5 } } { \exp_args:Nee \@@_Block_v:nnnnn }
7178          { \int_compare_p:nNn \l_tmpa_int = \c_one_int }
7179                                      { \exp_args:Nee \@@_Block_iv:nnnnn }
7180          { \int_compare_p:nNn \l_tmpb_int = \c_one_int }
7181                                      { \exp_args:Nee \@@_Block_iv:nnnnn }
7182        }
7183        { \exp_args:Nee \@@_Block_v:nnnnn }
7184      { \l_tmpa_int } { \l_tmpb_int } { #3 } { #4 } { #5 }
7185    }
```

The following macro is for the case of a `\Block` which is mono-row or mono-column (or both) and don't use the key p. In that case, the content of the block is composed right now in a box (because we have to take into account the dimensions of that box for the width of the current column or the height and the depth of the current row). However, that box will be put in the array *after the construction of the array* (by using PGF) with `\@@_draw_blocks:` and above all `\@@_Block_v:nnnnn` which will do the main job.
#1 is *i* (the number of rows of the block), #2 is *j* (the number of columns of the block), #3 is the list of *key=values* pairs, #4 are the tokens to put before the potential math mode and before the composition of the block and #5 is the label (=content) of the block.

```
7186  \cs_new_protected:Npn \@@_Block_iv:nnnnn #1 #2 #3 #4 #5
```

167

```
7187  {
7188    \int_gincr:N \g_@@_block_box_int
7189    \cs_set_protected_nopar:Npn \diagbox ##1 ##2
7190      {
7191        \tl_gput_right:Ne \g_@@_pre_code_after_tl
7192          {
7193            \@@_actually_diagbox:nnnnnn
7194              { \int_use:N \c@iRow }
7195              { \int_use:N \c@jCol }
7196              { \int_eval:n { \c@iRow + #1 - 1 } }
7197              { \int_eval:n { \c@jCol + #2 - 1 } }
7198              { \g_@@_row_style_tl \exp_not:n { ##1 } }
7199              { \g_@@_row_style_tl \exp_not:n { ##2 } }
7200          }
7201        }
7202    \box_gclear_new:c
7203      { 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!

```
7204    \hbox_gset:cn
7205      { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7206      {
```

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}).

```
7207        \tl_if_empty:NTF \l_@@_color_tl
7208          { \int_compare:nNnT { #2 } = \c_one_int \set@color }
7209          { \@@_color:o \l_@@_color_tl }
```

If the block is mono-row, we use \g_@@_row_style_tl even if it has yet been used in the beginning of the cell where the command \Block has been issued because we want to be able to take into account a potential instruction of color of the font in \g_@@_row_style_tl.

```
7210        \int_compare:nNnT { #1 } = \c_one_int
7211          {
7212            \int_if_zero:nTF \c@iRow
7213              {
```

In the following code, the value of `code-for-first-row` contains a \Block (in order to have the "first row" centered). But, that block will be executed, since it is entirely contained in the first row, the value of `code-for-first-row` will be inserted once again... with the same command \Block. That's why we have to nullify the command \Block.

```
$\begin{bNiceMatrix}%
  [
    r,
    first-row,
    last-col,
    code-for-first-row = \Block{}{\scriptstyle\color{blue} \arabic{jCol}},
    code-for-last-col = \scriptstyle \color{blue} \arabic{iRow}
  ]
    &  &  &  & \\
 -2 & 3 & -4 & 5 & \\
  3 & -4 & 5 & -6 & \\
 -4 & 5 & -6 & 7 & \\
  5 & -6 & 7 & -8 & \\
\end{bNiceMatrix}$
```

168

```
7214              \cs_set_eq:NN \Block \@@_NullBlock:
7215              \l_@@_code_for_first_row_tl
7216            }
7217            {
7218              \int_compare:nNnT \c@iRow = \l_@@_last_row_int
7219                {
7220                  \cs_set_eq:NN \Block \@@_NullBlock:
7221                  \l_@@_code_for_last_row_tl
7222                }
7223            }
7224          \g_@@_row_style_tl
7225        }
```

The following command will be no-op when `respect-arraystretch` is in force.

```
7226          \@@_reset_arraystretch:
7227          \dim_zero:N \extrarowheight
```

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

```
7228          #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.).

```
7229          \@@_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}.

```
7230          \bool_if:NTF \l_@@_tabular_bool
7231            {
7232              \bool_lazy_all:nTF
7233                {
7234                  { \int_compare_p:nNn { #2 } = \c_one_int }
```

Remind that, when the column has not a fixed width, the dimension `\l_@@_col_width_dim` has the conventional value of −1 cm.

```
7235                  { ! \dim_compare_p:nNn \l_@@_col_width_dim < \c_zero_dim }
7236                  { ! \g_@@_rotate_bool }
7237                }
```

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

```
7238                {
7239                  \use:e
7240                    {
7241                      \exp_not:N \begin { minipage }%
7242                        [ \str_lowercase:o \l_@@_vpos_block_str ]
7243                        { \l_@@_col_width_dim }
7244                      \str_case:on \l_@@_hpos_block_str
7245                        { c \centering r \raggedleft l \raggedright }
7246                    }
7247                  #5
7248                \end { minipage }
7249                }
```

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

```
7250                {
7251                  \use:e
7252                    {
7253                      \exp_not:N \begin { tabular }%
7254                        [ \str_lowercase:o \l_@@_vpos_block_str ]
7255                        { @ { } \l_@@_hpos_block_str @ { } }
7256                    }
7257                  #5
7258                \end { tabular }
```

```
7259                }
7260              }
```

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

```
7261            {
7262              \c_math_toggle_token
7263              \use:e
7264                {
7265                  \exp_not:N \begin { array }%
7266                    [ \str_lowercase:o \l_@@_vpos_block_str ]
7267                    { @ { } \l_@@_hpos_block_str @ { } }
7268                }
7269                #5
7270              \end { array }
7271              \c_math_toggle_token
7272            }
7273        }
```

The box which will contain the content of the block has now been composed.

If there were `\rotate` (which raises `\g_@@_rotate_bool`) in the content of the `\Block`, we do a rotation of the box (and we also adjust the baseline of the rotated box).

```
7274        \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.

```
7275        \int_compare:nNnT { #2 } = \c_one_int
7276          {
7277            \dim_gset:Nn \g_@@_blocks_wd_dim
7278              {
7279                \dim_max:nn
7280                  \g_@@_blocks_wd_dim
7281                  {
7282                    \box_wd:c
7283                      { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7284                  }
7285              }
7286          }
```

If we are in a mono-row block we take into account the height and the depth of that block for the height and the depth of the row, excepted when the block uses explicitly an option of vertical position.

```
7287        \bool_lazy_and:nnT
7288          { \int_compare_p:nNn { #1 } = \c_one_int }
```

If the user has not used a key for the vertical position of the block, then `\l_@@_vpos_block_str` remains empty.

```
7289          { \str_if_empty_p:N \l_@@_vpos_block_str }
7290          {
7291            \dim_gset:Nn \g_@@_blocks_ht_dim
7292              {
7293                \dim_max:nn
7294                  \g_@@_blocks_ht_dim
7295                  {
7296                    \box_ht:c
7297                      { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7298                  }
7299              }
7300            \dim_gset:Nn \g_@@_blocks_dp_dim
7301              {
7302                \dim_max:nn
7303                  \g_@@_blocks_dp_dim
7304                  {
7305                    \box_dp:c
```

```
7306                    { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7307                  }
7308              }
7309          }
7310      \seq_gput_right:Ne \g_@@_blocks_seq
7311          {
7312            \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.

```
7313          {
7314            \exp_not:n { #3 } ,
7315            \l_@@_hpos_block_str ,
```

Now, we put a key for the vertical alignment.

```
7316            \bool_if:NT \g_@@_rotate_bool
7317              {
7318                \bool_if:NTF \g_@@_rotate_c_bool
7319                  { m }
7320                  { \int_compare:nNnT \c@iRow = \l_@@_last_row_int T }
7321              }
7322          }
7323          {
7324            \box_use_drop:c
7325              { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7326          }
7327        }
7328      \bool_set_false:N \g_@@_rotate_c_bool
7329    }


7330 \cs_new:Npn \@@_adjust_hpos_rotate:
7331    {
7332      \bool_if:NT \g_@@_rotate_bool
7333        {
7334          \str_set:Ne \l_@@_hpos_block_str
7335            {
7336              \bool_if:NTF \g_@@_rotate_c_bool
7337                { c }
7338                {
7339                  \str_case:onF \l_@@_vpos_block_str
7340                    { b l B l t r T r }
7341                    { \int_compare:nNnTF \c@iRow = \l_@@_last_row_int r l }
7342                }
7343            }
7344        }
7345    }
```

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

```
7346 \cs_new_protected:Npn \@@_rotate_box_of_block:
7347    {
7348      \box_grotate:cn
7349        { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7350        { 90 }
7351      \int_compare:nNnT \c@iRow = \l_@@_last_row_int
7352        {
7353          \vbox_gset_top:cn
7354            { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7355            {
7356              \skip_vertical:n { 0.8 ex }
```

```
7357            \box_use:c
7358              { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7359          }
7360        }
7361      \bool_if:NT \g_@@_rotate_c_bool
7362        {
7363          \hbox_gset:cn
7364            { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7365            {
7366              \c_math_toggle_token
7367              \vcenter
7368                {
7369                  \box_use:c
7370                  { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7371                }
7372              \c_math_toggle_token
7373            }
7374        }
7375    }
```

The following macro is for the standard case, where the block is not mono-row and not mono-column and does not use the key p). In that case, the content of the block is *not* composed right now in a box. The composition in a box will be done further, just after the construction of the array (cf. `\@@_draw_blocks:` and above all `\@@_Block_v:nnnnn`).
#1 is i (the number of rows of the block), #2 is j (the number of columns of the block), #3 is the list of *key=values* pairs, #4 are the tokens to put before the math mode and before the composition of the block and #5 is the label (=content) of the block.

```
7376  \cs_new_protected:Npn \@@_Block_v:nnnnn #1 #2 #3 #4 #5
7377    {
7378      \seq_gput_right:Ne \g_@@_blocks_seq
7379        {
7380          \l_tmpa_tl
7381          { \exp_not:n { #3 } }
7382          {
7383            \bool_if:NTF \l_@@_tabular_bool
7384              {
7385                \group_begin:
```

The following command will be no-op when `respect-arraystretch` is in force.

```
7386                \@@_reset_arraystretch:
7387                \exp_not:n
7388                  {
7389                    \dim_zero:N \extrarowheight
7390                    #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).

```
7391                    \bool_if:NT \c_@@_testphase_table_bool
7392                      { \tag_stop:n { table } }
7393                    \use:e
7394                      {
7395                        \exp_not:N \begin { tabular } [ \l_@@_vpos_block_str ]
7396                        { @ { } \l_@@_hpos_block_str @ { } }
7397                      }
7398                    #5
7399                    \end { tabular }
7400                  }
7401                \group_end:
7402              }
```

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

```
7403                 {
7404                     \group_begin:
```

The following will be no-op when respect-arraystretch is in force.

```
7405                     \@@_reset_arraystretch:
7406                     \exp_not:n
7407                       {
7408                         \dim_zero:N \extrarowheight
7409                         #4
7410                         \c_math_toggle_token
7411                         \use:e
7412                           {
7413                             \exp_not:N \begin { array } [ \l_@@_vpos_block_str ]
7414                             { @ { } \l_@@_hpos_block_str @ { } }
7415                           }
7416                         #5
7417                         \end { array }
7418                         \c_math_toggle_token
7419                       }
7420                     \group_end:
7421                   }
7422               }
7423           }
7424     }
```

The following macro is for the case of a \Block which uses the key p.

```
7425 \cs_new_protected:Npn \@@_Block_vi:nnnnn #1 #2 #3 #4 #5
7426   {
7427     \seq_gput_right:Ne \g_@@_blocks_seq
7428       {
7429         \l_tmpa_tl
7430         { \exp_not:n { #3 } }
7431         {
7432           \group_begin:
7433           \exp_not:n { #4 #5 }
7434           \group_end:
7435         }
7436       }
7437   }
```

The following macro is for the case of a \Block which uses the key p.

```
7438 \cs_new_protected:Npn \@@_Block_vii:nnnnn #1 #2 #3 #4 #5
7439   {
7440     \seq_gput_right:Ne \g_@@_blocks_seq
7441       {
7442         \l_tmpa_tl
7443         { \exp_not:n { #3 } }
7444         { \exp_not:n { #4 #5 } }
7445       }
7446   }
```

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).

```
7447 \keys_define:nn { nicematrix / Block / SecondPass }
7448   {
7449     ampersand-in-blocks .bool_set:N = \l_@@_amp_in_blocks_bool ,
7450     ampersand-in-blocks .default:n = true ,
7451     &-in-blocks .meta:n = ampersand-in-blocks ,
7452     tikz .code:n =
```

```
7453        \IfPackageLoadedTF { tikz }
7454          { \seq_put_right:Nn \l_@@_tikz_seq { { #1 } } }
7455          { \@@_error:n { tikz~key~without~tikz } } ,
7456      tikz .value_required:n = true ,
7457      fill .code:n =
7458        \tl_set_rescan:Nnn
7459          \l_@@_fill_tl
7460          { \char_set_catcode_other:N ! }
7461          { #1 } ,
7462      fill .value_required:n = true ,
7463      opacity .tl_set:N = \l_@@_opacity_tl ,
7464      opacity .value_required:n = true ,
7465      draw .code:n =
7466        \tl_set_rescan:Nnn
7467          \l_@@_draw_tl
7468          { \char_set_catcode_other:N ! }
7469          { #1 } ,
7470      draw .default:n = default ,
7471      rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
7472      rounded-corners .default:n = 4 pt ,
7473      color .code:n =
7474        \@@_color:n { #1 }
7475        \tl_set_rescan:Nnn
7476          \l_@@_draw_tl
7477          { \char_set_catcode_other:N ! }
7478          { #1 } ,
7479      borders .clist_set:N = \l_@@_borders_clist ,
7480      borders .value_required:n = true ,
7481      hvlines .meta:n = { vlines , hlines } ,
7482      vlines .bool_set:N = \l_@@_vlines_block_bool,
7483      vlines .default:n = true ,
7484      hlines .bool_set:N = \l_@@_hlines_block_bool,
7485      hlines .default:n = true ,
7486      line-width .dim_set:N = \l_@@_line_width_dim ,
7487      line-width .value_required:n = true ,
```

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

```
7488      j .code:n = \str_set:Nn \l_@@_hpos_block_str j
7489                  \bool_set_true:N \l_@@_p_block_bool ,
7490      l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7491      r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7492      c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7493      L .code:n = \str_set:Nn \l_@@_hpos_block_str l
7494                  \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7495      R .code:n = \str_set:Nn \l_@@_hpos_block_str r
7496                  \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7497      C .code:n = \str_set:Nn \l_@@_hpos_block_str c
7498                  \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7499      t .code:n = \str_set:Nn \l_@@_vpos_block_str t ,
7500      T .code:n = \str_set:Nn \l_@@_vpos_block_str T ,
7501      b .code:n = \str_set:Nn \l_@@_vpos_block_str b ,
7502      B .code:n = \str_set:Nn \l_@@_vpos_block_str B ,
7503      m .code:n = \str_set:Nn \l_@@_vpos_block_str c ,
7504      m .value_forbidden:n = true ,
7505      v-center .meta:n = m ,
7506      p .code:n = \bool_set_true:N \l_@@_p_block_bool ,
7507      p .value_forbidden:n = true ,
7508      name .tl_set:N = \l_@@_block_name_str ,
7509      name .value_required:n = true ,
7510      name .initial:n = ,
7511      respect-arraystretch .code:n =
7512        \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
7513      respect-arraystretch .value_forbidden:n = true ,
7514      transparent .bool_set:N = \l_@@_transparent_bool ,
```

```
7515    transparent .default:n = true ,
7516    transparent .initial:n = false ,
7517    unknown .code:n = \@@_error:n { Unknown~key~for~Block }
7518  }
```

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.

```
7519 \cs_new_protected:Npn \@@_draw_blocks:
7520   {
7521     \bool_if:NTF \c_@@_tagging_array_bool
7522       { \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign: }
7523       { \cs_set_eq:NN \ialign \@@_old_ialign: }
7524     \seq_map_inline:Nn \g_@@_blocks_seq { \@@_Block_iv:nnnnnn ##1 }
7525   }

7526 \cs_new_protected:Npn \@@_Block_iv:nnnnnn #1 #2 #3 #4 #5 #6
7527   {
```

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

```
7528     \int_zero_new:N \l_@@_last_row_int
7529     \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.

```
7530     \int_compare:nNnTF { #3 } > { 99 }
7531       { \int_set_eq:NN \l_@@_last_row_int \c@iRow }
7532       { \int_set:Nn \l_@@_last_row_int { #3 } }
7533     \int_compare:nNnTF { #4 } > { 99 }
7534       { \int_set_eq:NN \l_@@_last_col_int \c@jCol }
7535       { \int_set:Nn \l_@@_last_col_int { #4 } }
7536     \int_compare:nNnTF \l_@@_last_col_int > \g_@@_col_total_int
7537       {
7538         \bool_lazy_and:nnTF
7539           \l_@@_preamble_bool
7540           {
7541             \int_compare_p:n
7542               { \l_@@_last_col_int <= \g_@@_static_num_of_col_int }
7543           }
7544           {
7545             \msg_error:nnnn { nicematrix } { Block~too~large~2 } { #1 } { #2 }
7546             \@@_msg_redirect_name:nn { Block~too~large~2 } { none }
7547             \@@_msg_redirect_name:nn { columns~not~used } { none }
7548           }
7549           { \msg_error:nnnn { nicematrix } { Block~too~large~1 } { #1 } { #2 } }
7550       }
7551       {
7552         \int_compare:nNnTF \l_@@_last_row_int > \g_@@_row_total_int
7553           { \msg_error:nnnn { nicematrix } { Block~too~large~1 } { #1 } { #2 } }
7554           {
7555             \@@_Block_v:nnVVnn
7556               { #1 }
7557               { #2 }
7558               \l_@@_last_row_int
7559               \l_@@_last_col_int
7560               { #5 }
7561               { #6 }
7562           }
```

```
7563          }
7564      }
```

```
7565  \cs_generate_variant:Nn \@@_Block_v:nnnnnn { n n V V n n }
```

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

```
7566  \cs_new_protected:Npn \@@_Block_v:nnnnnn #1 #2 #3 #4 #5 #6
7567      {
```

The group is for the keys.

```
7568      \group_begin:
7569      \int_compare:nNnT { #1 } = { #3 }
7570        { \str_set:Nn \l_@@_vpos_block_str { t } }
7571      \keys_set:nn { nicematrix / Block / SecondPass } { #5 }
```

If the content of the block contains &, we will have a special treatement (since the cell must be divided in several sub-cells).

```
7572      \tl_if_in:nnT { #6 } { & } { \bool_set_true:N \l_@@_ampersand_bool }
7573      \bool_lazy_and:nnT
7574        \l_@@_vlines_block_bool
7575        { ! \l_@@_ampersand_bool }
7576        {
7577          \tl_gput_right:Ne \g_nicematrix_code_after_tl
7578            {
7579              \@@_vlines_block:nnn
7580                { \exp_not:n { #5 } }
7581                { #1 - #2 }
7582                { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7583            }
7584        }
7585      \bool_if:NT \l_@@_hlines_block_bool
7586        {
7587          \tl_gput_right:Ne \g_nicematrix_code_after_tl
7588            {
7589              \@@_hlines_block:nnn
7590                { \exp_not:n { #5 } }
7591                { #1 - #2 }
7592                { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7593            }
7594        }
7595      \bool_if:NF \l_@@_transparent_bool
7596        {
7597          \bool_lazy_and:nnF \l_@@_vlines_block_bool \l_@@_hlines_block_bool
7598            {
```

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).

```
7599              \seq_gput_left:Ne \g_@@_pos_of_blocks_seq
7600                { { #1 } { #2 } { #3 } { #4 } { \l_@@_block_name_str } }
7601            }
7602        }


7603      \tl_if_empty:NF \l_@@_draw_tl
7604        {
7605          \bool_lazy_or:nnT \l_@@_hlines_block_bool \l_@@_vlines_block_bool
7606            { \@@_error:n { hlines~with~color } }
7607          \tl_gput_right:Ne \g_nicematrix_code_after_tl
7608            {
7609              \@@_stroke_block:nnn
```

#5 are the options

```
7610                 { \exp_not:n { #5 } }
7611                 { #1 - #2 }
7612                 { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7613             }
7614         \seq_gput_right:Nn \g_@@_pos_of_stroken_blocks_seq
7615             { { #1 } { #2 } { #3 } { #4 } }
7616       }
7617     \clist_if_empty:NF \l_@@_borders_clist
7618       {
7619         \tl_gput_right:Ne \g_nicematrix_code_after_tl
7620           {
7621             \@@_stroke_borders_block:nnn
7622               { \exp_not:n { #5 } }
7623               { #1 - #2 }
7624               { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7625           }
7626       }
7627     \tl_if_empty:NF \l_@@_fill_tl
7628       {
7629         \tl_if_empty:NF \l_@@_opacity_tl
7630           {
7631             \tl_if_head_eq_meaning:nNTF \l_@@_fill_tl [
7632               {
7633                 \tl_set:Ne \l_@@_fill_tl
7634                   {
7635                     [ opacity = \l_@@_opacity_tl ,
7636                     \tl_tail:o \l_@@_fill_tl
7637                   }
7638               }
7639               {
7640                 \tl_set:Ne \l_@@_fill_tl
7641                   { [ opacity = \l_@@_opacity_tl ] { \l_@@_fill_tl } }
7642               }
7643           }
7644         \tl_gput_right:Ne \g_@@_pre_code_before_tl
7645           {
7646             \exp_not:N \roundedrectanglecolor
7647               \exp_args:No \tl_if_head_eq_meaning:nNTF \l_@@_fill_tl [
7648                 { \l_@@_fill_tl }
7649                 { { \l_@@_fill_tl } }
7650               { #1 - #2 }
7651               { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7652               { \dim_use:N \l_@@_rounded_corners_dim }
7653           }
7654       }
7655     \seq_if_empty:NF \l_@@_tikz_seq
7656       {
7657         \tl_gput_right:Ne \g_nicematrix_code_before_tl
7658           {
7659             \@@_block_tikz:nnnnn
7660               { #1 }
7661               { #2 }
7662               { \int_use:N \l_@@_last_row_int }
7663               { \int_use:N \l_@@_last_col_int }
7664               { \seq_use:Nn \l_@@_tikz_seq { , } }
7665           }
7666       }


7667     \cs_set_protected_nopar:Npn \diagbox ##1 ##2
7668       {
7669         \tl_gput_right:Ne \g_@@_pre_code_after_tl
```

177

```
7670            {
7671              \@@_actually_diagbox:nnnnnn
7672                { #1 }
7673                { #2 }
7674                { \int_use:N \l_@@_last_row_int }
7675                { \int_use:N \l_@@_last_col_int }
7676                { \exp_not:n { ##1 } } { \exp_not:n { ##2 } } }
7677            }
7678          }
```

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   We highlight the node 1-1-block-short

The construction of the node corresponding to the merged cells.

```
7679        \pgfpicture
7680        \pgfrememberpicturepositiononpagetrue
7681        \pgf@relevantforpicturesizefalse
7682        \@@_qpoint:n { row - #1 }
7683        \dim_set_eq:NN \l_tmpa_dim \pgf@y
7684        \@@_qpoint:n { col - #2 }
7685        \dim_set_eq:NN \l_tmpb_dim \pgf@x
7686        \@@_qpoint:n { row - \int_eval:n { \l_@@_last_row_int + 1 } }
7687        \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7688        \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7689        \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.

```
7690        \@@_pgf_rect_node:nnnnn
7691          { \@@_env: - #1 - #2 - block }
7692          \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
7693        \str_if_empty:NF \l_@@_block_name_str
7694          {
7695            \pgfnodealias
7696              { \@@_env: - \l_@@_block_name_str }
7697              { \@@_env: - #1 - #2 - block }
7698            \str_if_empty:NF \l_@@_name_str
7699              {
7700                \pgfnodealias
7701                  { \l_@@_name_str - \l_@@_block_name_str }
7702                  { \@@_env: - #1 - #2 - block }
7703              }
7704          }
```

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.

```
7705        \bool_if:NF \l_@@_hpos_of_block_cap_bool
7706          {
7707            \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.

```
7708            \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
7709              {
```

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.

```
7710                \cs_if_exist:cT
7711                  { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
7712                  {
7713                    \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
7714                      {
7715                        \pgfpointanchor { \@@_env: - ##1 - #2 } { west }
7716                        \dim_set:Nn \l_tmpb_dim { \dim_min:nn \l_tmpb_dim \pgf@x }
7717                      }
7718                  }
7719              }
```

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.

```
7720            \dim_compare:nNnT \l_tmpb_dim = \c_max_dim
7721              {
7722                \@@_qpoint:n { col - #2 }
7723                \dim_set_eq:NN \l_tmpb_dim \pgf@x
7724              }
7725            \dim_set:Nn \l_@@_tmpd_dim { - \c_max_dim }
7726            \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
7727              {
7728                \cs_if_exist:cT
7729                  { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
7730                  {
7731                    \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
7732                      {
7733                        \pgfpointanchor
7734                          { \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
7735                          { east }
7736                        \dim_set:Nn \l_@@_tmpd_dim { \dim_max:nn \l_@@_tmpd_dim \pgf@x }
7737                      }
7738                  }
7739              }
7740            \dim_compare:nNnT \l_@@_tmpd_dim = { - \c_max_dim }
7741              {
7742                \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7743                \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
7744              }
7745            \@@_pgf_rect_node:nnnnn
7746              { \@@_env: - #1 - #2 - block - short }
7747              \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
7748          }
```

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.

```
7749        \bool_if:NT \l_@@_medium_nodes_bool
7750          {
7751            \@@_pgf_rect_node:nnn
```

```
7752            { \@@_env: - #1 - #2 - block - medium }
7753            { \pgfpointanchor { \@@_env: - #1 - #2 - medium } { north~west } }
7754            {
7755              \pgfpointanchor
7756                { \@@_env:
7757                  - \int_use:N \l_@@_last_row_int
7758                  - \int_use:N \l_@@_last_col_int - medium
7759                }
7760                { south~east }
7761            }
7762        }
7763      \endpgfpicture


7764    \bool_if:NTF \l_@@_ampersand_bool
7765      {
7766        \seq_set_split:Nnn \l_tmpa_seq { & } { #6 }
7767        \int_zero_new:N \l_@@_split_int
7768        \int_set:Nn \l_@@_split_int { \seq_count:N \l_tmpa_seq }
7769        \pgfpicture
7770        \pgfrememberpicturepositiononpagetrue
7771        \pgf@relevantforpicturesizefalse
7772        \@@_qpoint:n { row - #1 }
7773        \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7774        \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
7775        \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
7776        \@@_qpoint:n { col - #2 }
7777        \dim_set_eq:NN \l_tmpa_dim \pgf@x
7778        \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
7779        \dim_set:Nn \l_tmpb_dim
7780          { ( \pgf@x - \l_tmpa_dim ) / \int_use:N \l_@@_split_int }
7781        \bool_lazy_or:nnT
7782          \l_@@_vlines_block_bool
7783          { \tl_if_eq_p:NN \l_@@_vlines_clist \c_@@_all_tl }
7784          {
7785            \int_step_inline:nn { \l_@@_split_int - 1 }
7786              {
7787                \pgfpathmoveto
7788                  {
7789                    \pgfpoint
7790                      { \l_tmpa_dim + ##1 \l_tmpb_dim }
7791                      \l_@@_tmpc_dim
7792                  }
7793                \pgfpathlineto
7794                  {
7795                    \pgfpoint
7796                      { \l_tmpa_dim + ##1 \l_tmpb_dim }
7797                      \l_@@_tmpd_dim
7798                  }
7799                \CT@arc@
7800                \pgfsetlinewidth { 1.1 \arrayrulewidth }
7801                \pgfsetrectcap
7802                \pgfusepathqstroke
7803              }
7804          }
7805        \@@_qpoint:n { row - #1 - base }
7806        \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7807        \int_step_inline:nn \l_@@_split_int
7808          {
7809            \group_begin:
7810            \dim_set:Nn \col@sep
7811              { \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
7812            \pgftransformshift
7813              {
```

```
7814                \pgfpoint
7815                  {
7816                    \str_case:on \l_@@_hpos_block_str
7817                      {
7818                        l { \l_tmpa_dim + ##1 \l_tmpb_dim - \l_tmpb_dim + \col@sep}
7819                        c { \l_tmpa_dim + ##1 \l_tmpb_dim - 0.5 \l_tmpb_dim }
7820                        r { \l_tmpa_dim + ##1 \l_tmpb_dim - \col@sep }
7821                      }
7822                  }
7823                  { \l_@@_tmpc_dim }
7824              }
7825          \pgfset
7826            {
7827              inner~xsep = \c_zero_dim ,
7828              inner~ysep = \c_zero_dim
7829            }
7830          \pgfnode
7831            { rectangle }
7832            {
7833              \str_case:on \l_@@_hpos_block_str
7834                {
7835                  c { base }
7836                  l { base~west }
7837                  r { base~east }
7838                }
7839            }
7840            { \seq_item:Nn \l_tmpa_seq { ##1 } } { } { }
7841          \group_end:
7842        }
7843      \endpgfpicture
7844    }
```

Now the case where there is no ampersand & in the content of the block.

```
7845      {
7846        \bool_if:NTF \l_@@_p_block_bool
7847          {
```

When the final user has used the key p, we have to compute the width.

```
7848            \pgfpicture
7849              \pgfremeberpicturepositiononpagetrue
7850              \pgf@relevantforpicturesizefalse
7851              \bool_if:NTF \l_@@_hpos_of_block_cap_bool
7852                {
7853                  \@@_qpoint:n { col - #2 }
7854                  \dim_gset_eq:NN \g_tmpa_dim \pgf@x
7855                  \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7856                }
7857                {
7858                  \pgfpointanchor { \@@_env: - #1 - #2 - block - short } { west }
7859                  \dim_gset_eq:NN \g_tmpa_dim \pgf@x
7860                  \pgfpointanchor { \@@_env: - #1 - #2 - block - short } { east }
7861                }
7862              \dim_gset:Nn \g_tmpb_dim { \pgf@x - \g_tmpa_dim }
7863            \endpgfpicture
7864            \hbox_set:Nn \l_@@_cell_box
7865              {
7866                \begin { minipage } [ \str_lowercase:o \l_@@_vpos_block_str ]
7867                  { \g_tmpb_dim }
7868                \str_case:on \l_@@_hpos_block_str
7869                  { c \centering r \raggedleft l \raggedright j { } }
7870                #6
7871                \end { minipage }
7872              }
7873          }
```

```
7874              { \hbox_set:Nn \l_@@_cell_box { \set@color #6 } }
7875          \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
```

Now, we will put the label of the block. We recall that `\l_@@_vpos_block_str` is empty when the user has not used a key for the vertical position of the block.

```
7876          \pgfpicture
7877          \pgfrememberpicturepositiononpagetrue
7878          \pgf@relevantforpicturesizefalse
7879          \bool_lazy_any:nTF
7880            {
7881              { \str_if_empty_p:N \l_@@_vpos_block_str } % added 2024/06/29
7882              { \str_if_eq_p:on \l_@@_vpos_block_str { c } }
7883              { \str_if_eq_p:on \l_@@_vpos_block_str { T } }
7884              { \str_if_eq_p:on \l_@@_vpos_block_str { B } }
7885            }

7886            {
```

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

```
7887                \int_if_zero:nT { #2 } { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_r_str }
```

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

```
7888                \bool_if:nT \g_@@_last_col_found_bool
7889                  {
7890                    \int_compare:nNnT { #2 } = \g_@@_col_total_int
7891                      { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_l_str }
7892                  }
```

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

```
7893                \tl_set:Ne \l_tmpa_tl
7894                  {
7895                    \str_case:on \l_@@_vpos_block_str
7896                      {
```

We recall that `\l_@@_vpos_block_str` is empty when the user has not used a key for the vertical position of the block.

```
7897                        { } { % added 2024-06-29
7898                            \str_case:on \l_@@_hpos_block_str
7899                              {
7900                                c { center }
7901                                l { west }
7902                                r { east }
7903                                j { center }
7904                              }
7905                          }
7906                        c {
7907                            \str_case:on \l_@@_hpos_block_str
7908                              {
7909                                c { center }
7910                                l { west }
7911                                r { east }
7912                                j { center }
7913                              }
7914
7915                          }
7916                        T {
7917                            \str_case:on \l_@@_hpos_block_str
7918                              {
7919                                c { north }
7920                                l { north~west }
7921                                r { north~east }
7922                                j { north }
7923                              }
7924
```

182

```
7925                              }
7926                         B {
7927                             \str_case:on \l_@@_hpos_block_str
7928                                {
7929                                   c { south }
7930                                   l { south~west }
7931                                   r { south~east }
7932                                   j { south }
7933                                }

7934
7935                         }
7936                     }
7937                 }
7938            \pgftransformshift
7939              {
7940                 \pgfpointanchor
7941                   {
7942                      \@@_env: - #1 - #2 - block
7943                      \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
7944                   }
7945                   { \l_tmpa_tl }
7946              }
7947            \pgfset
7948              {
7949                 inner~xsep = \c_zero_dim ,
7950                 inner~ysep = \c_zero_dim
7951              }
7952            \pgfnode
7953              { rectangle }
7954              { \l_tmpa_tl }
7955              { \box_use_drop:N \l_@@_cell_box } { } { }
7956         }
```

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

```
7957             {
7958                \pgfextracty \l_tmpa_dim
7959                  {
7960                     \@@_qpoint:n
7961                       {
7962                          row - \str_if_eq:onTF \l_@@_vpos_block_str { b } { #3 } { #1 }
7963                          - base
7964                       }
7965                  }
7966                \dim_sub:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }
```

We retrieve (in `\pgf@x`) the *x*-value of the center of the block.

```
7967                \pgfpointanchor
7968                  {
7969                     \@@_env: - #1 - #2 - block
7970                     \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
7971                  }
7972                  {
7973                     \str_case:on \l_@@_hpos_block_str
7974                       {
7975                          c { center }
7976                          l { west }
7977                          r { east }
7978                          j { center }
7979                       }
7980                  }
```

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

```
7981                \pgftransformshift { \pgfpoint \pgf@x \l_tmpa_dim }
7982                \pgfset { inner~sep = \c_zero_dim }
```

183

```
7983            \pgfnode
7984              { rectangle }
7985              {
7986                 \str_case:on \l_@@_hpos_block_str
7987                   {
7988                      c { base }
7989                      l { base~west }
7990                      r { base~east }
7991                      j { base }
7992                   }
7993              }
7994              { \box_use_drop:N \l_@@_cell_box } { } { }
7995            }
7996          \endpgfpicture
7997        }
7998      \group_end:
7999    }
```

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).

```
8000 \cs_new_protected:Npn \@@_stroke_block:nnn #1 #2 #3
8001   {
8002     \group_begin:
8003     \tl_clear:N \l_@@_draw_tl
8004     \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8005     \keys_set_known:nn { nicematrix / BlockStroke } { #1 }
8006     \pgfpicture
8007     \pgfrememberpicturepositiononpagetrue
8008     \pgf@relevantforpicturesizefalse
8009     \tl_if_empty:NF \l_@@_draw_tl
8010       {
```

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

```
8011          \tl_if_eq:NNTF \l_@@_draw_tl \c_@@_default_tl
8012            { \CT@arc@ }
8013            { \@@_color:o \l_@@_draw_tl }
8014       }
8015     \pgfsetcornersarced
8016       {
8017          \pgfpoint
8018          { \l_@@_rounded_corners_dim }
8019          { \l_@@_rounded_corners_dim }
8020       }
8021     \@@_cut_on_hyphen:w #2 \q_stop
8022     \int_compare:nNnF \l_tmpa_tl > \c@iRow
8023       {
8024          \int_compare:nNnF \l_tmpb_tl > \c@jCol
8025            {
8026               \@@_qpoint:n { row - \l_tmpa_tl }
8027               \dim_set_eq:NN \l_tmpb_dim \pgf@y
8028               \@@_qpoint:n { col - \l_tmpb_tl }
8029               \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
8030               \@@_cut_on_hyphen:w #3 \q_stop
8031               \int_compare:nNnT \l_tmpa_tl > \c@iRow
8032                 { \tl_set:No \l_tmpa_tl { \int_use:N \c@iRow } }
8033               \int_compare:nNnT \l_tmpb_tl > \c@jCol
8034                 { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
8035               \@@_qpoint:n { row - \int_eval:n { \l_tmpa_tl + 1 } }
8036               \dim_set_eq:NN \l_tmpa_dim \pgf@y
8037               \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
8038               \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
```

184

```
8039                \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
8040                \pgfpathrectanglecorners
8041                  { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
8042                  { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
8043                \dim_compare:nNnTF \l_@@_rounded_corners_dim = \c_zero_dim
8044                  { \pgfusepathqstroke }
8045                  { \pgfusepath { stroke } }
8046              }
8047          }
8048        \endpgfpicture
8049        \group_end:
8050    }
```

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

```
8051  \keys_define:nn { nicematrix / BlockStroke }
8052    {
8053      color .tl_set:N = \l_@@_draw_tl ,
8054      draw .code:n =
8055        \exp_args:Ne \tl_if_empty:nF { #1 } { \tl_set:Nn \l_@@_draw_tl { #1 } } ,
8056      draw .default:n = default ,
8057      line-width .dim_set:N = \l_@@_line_width_dim ,
8058      rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
8059      rounded-corners .default:n = 4 pt
8060    }
```

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).

```
8061  \cs_new_protected:Npn \@@_vlines_block:nnn #1 #2 #3
8062    {
8063      \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8064      \keys_set_known:nn { nicematrix / BlockBorders } { #1 }
8065      \@@_cut_on_hyphen:w #2 \q_stop
8066      \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
8067      \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
8068      \@@_cut_on_hyphen:w #3 \q_stop
8069      \tl_set:Ne \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
8070      \tl_set:Ne \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
8071      \int_step_inline:nnn \l_@@_tmpd_tl \l_tmpb_tl
8072        {
8073          \use:e
8074            {
8075              \@@_vline:n
8076                {
8077                  position = ##1 ,
8078                  start = \l_@@_tmpc_tl ,
8079                  end = \int_eval:n { \l_tmpa_tl - 1 } ,
8080                  total-width = \dim_use:N \l_@@_line_width_dim
8081                }
8082            }
8083        }
8084    }
8085  \cs_new_protected:Npn \@@_hlines_block:nnn #1 #2 #3
8086    {
8087      \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8088      \keys_set_known:nn { nicematrix / BlockBorders } { #1 }
8089      \@@_cut_on_hyphen:w #2 \q_stop
8090      \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
8091      \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
8092      \@@_cut_on_hyphen:w #3 \q_stop
8093      \tl_set:Ne \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
8094      \tl_set:Ne \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
8095      \int_step_inline:nnn \l_@@_tmpc_tl \l_tmpa_tl
```

```
8096          {
8097            \use:e
8098              {
8099                \@@_hline:n
8100                  {
8101                    position = ##1 ,
8102                    start = \l_@@_tmpd_tl ,
8103                    end = \int_eval:n { \l_tmpb_tl - 1 } ,
8104                    total-width = \dim_use:N \l_@@_line_width_dim
8105                  }
8106              }
8107          }
8108    }
```

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).

```
8109  \cs_new_protected:Npn \@@_stroke_borders_block:nnn #1 #2 #3
8110    {
8111      \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8112      \keys_set_known:nn { nicematrix / BlockBorders } { #1 }
8113      \dim_compare:nNnTF \l_@@_rounded_corners_dim > \c_zero_dim
8114        { \@@_error:n { borders~forbidden } }
8115        {
8116          \tl_clear_new:N \l_@@_borders_tikz_tl
8117          \keys_set:nV
8118            { nicematrix / OnlyForTikzInBorders }
8119            \l_@@_borders_clist
8120          \@@_cut_on_hyphen:w #2 \q_stop
8121          \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
8122          \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
8123          \@@_cut_on_hyphen:w #3 \q_stop
8124          \tl_set:Ne \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
8125          \tl_set:Ne \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
8126          \@@_stroke_borders_block_i:
8127        }
8128    }
8129  \hook_gput_code:nnn { begindocument } { . }
8130    {
8131      \cs_new_protected:Npx \@@_stroke_borders_block_i:
8132        {
8133          \c_@@_pgfortikzpicture_tl
8134          \@@_stroke_borders_block_ii:
8135          \c_@@_endpgfortikzpicture_tl
8136        }
8137    }
8138  \cs_new_protected:Npn \@@_stroke_borders_block_ii:
8139    {
8140      \pgfrememberpicturepositiononpagetrue
8141      \pgf@relevantforpicturesizefalse
8142      \CT@arc@
8143      \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
8144      \clist_if_in:NnT \l_@@_borders_clist { right }
8145        { \@@_stroke_vertical:n \l_tmpb_tl }
8146      \clist_if_in:NnT \l_@@_borders_clist { left }
8147        { \@@_stroke_vertical:n \l_@@_tmpd_tl }
8148      \clist_if_in:NnT \l_@@_borders_clist { bottom }
8149        { \@@_stroke_horizontal:n \l_tmpa_tl }
8150      \clist_if_in:NnT \l_@@_borders_clist { top }
8151        { \@@_stroke_horizontal:n \l_@@_tmpc_tl }
8152    }
```

```
8153  \keys_define:nn { nicematrix / OnlyForTikzInBorders }
8154    {
8155      tikz .code:n =
8156        \cs_if_exist:NTF \tikzpicture
8157          { \tl_set:Nn \l_@@_borders_tikz_tl { #1 } }
8158          { \@@_error:n { tikz~in~borders~without~tikz } } ,
8159      tikz .value_required:n = true ,
8160      top .code:n = ,
8161      bottom .code:n = ,
8162      left .code:n = ,
8163      right .code:n = ,
8164      unknown .code:n = \@@_error:n { bad~border }
8165    }
```

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).

```
8166  \cs_new_protected:Npn \@@_stroke_vertical:n #1
8167    {
8168      \@@_qpoint:n \l_@@_tmpc_tl
8169      \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
8170      \@@_qpoint:n \l_tmpa_tl
8171      \dim_set:Nn \l_@@_tmpc_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
8172      \@@_qpoint:n { #1 }
8173      \tl_if_empty:NTF \l_@@_borders_tikz_tl
8174        {
8175          \pgfpathmoveto { \pgfpoint \pgf@x \l_tmpb_dim }
8176          \pgfpathlineto { \pgfpoint \pgf@x \l_@@_tmpc_dim }
8177          \pgfusepathqstroke
8178        }
8179        {
8180          \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
8181            ( \pgf@x , \l_tmpb_dim ) -- ( \pgf@x , \l_@@_tmpc_dim ) ;
8182        }
8183    }
```

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).

```
8184  \cs_new_protected:Npn \@@_stroke_horizontal:n #1
8185    {
8186      \@@_qpoint:n \l_@@_tmpd_tl
8187      \clist_if_in:NnTF \l_@@_borders_clist { left }
8188        { \dim_set:Nn \l_tmpa_dim { \pgf@x - 0.5 \l_@@_line_width_dim } }
8189        { \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \l_@@_line_width_dim } }
8190      \@@_qpoint:n \l_tmpb_tl
8191      \dim_set:Nn \l_tmpb_dim { \pgf@x + 0.5 \l_@@_line_width_dim }
8192      \@@_qpoint:n { #1 }
8193      \tl_if_empty:NTF \l_@@_borders_tikz_tl
8194        {
8195          \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }
8196          \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
8197          \pgfusepathqstroke
8198        }
8199        {
8200          \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
8201            ( \l_tmpa_dim , \pgf@y ) -- ( \l_tmpb_dim , \pgf@y ) ;
8202        }
8203    }
```

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

```
8204  \keys_define:nn { nicematrix / BlockBorders }
8205    {
8206      borders .clist_set:N = \l_@@_borders_clist ,
```

```
8207    rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
8208    rounded-corners .default:n = 4 pt ,
8209    line-width .dim_set:N = \l_@@_line_width_dim
8210  }
```

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.

```
8211  \cs_new_protected:Npn \@@_block_tikz:nnnnn #1 #2 #3 #4 #5
8212    {
8213      \begin { tikzpicture }
8214      \@@_clip_with_rounded_corners:
8215      \clist_map_inline:nn { #5 }
8216        {
8217          \keys_set_known:nnN { nicematrix / SpecialOffset } { ##1 } \l_tmpa_tl
8218          \use:e { \exp_not:N \path [ \l_tmpa_tl ] }
8219              (
8220                [
8221                  xshift = \dim_use:N \l_@@_offset_dim ,
8222                  yshift = - \dim_use:N \l_@@_offset_dim
8223                ]
8224                #1 -| #2
8225              )
8226              rectangle
8227              (
8228                [
8229                  xshift = - \dim_use:N \l_@@_offset_dim ,
8230                  yshift = \dim_use:N \l_@@_offset_dim
8231                ]
8232                \int_eval:n { #3 + 1 } -| \int_eval:n { #4 + 1 }
8233              ) ;
8234        }
8235      \end { tikzpicture }
8236    }
8237  \cs_generate_variant:Nn \@@_block_tikz:nnnnn { n n n n V }

8238  \keys_define:nn { nicematrix / SpecialOffset }
8239    { offset .dim_set:N = \l_@@_offset_dim }
```

In some circonstancies, we want to nullify the command `\Block`. In order to reach that goal, we will link the command `\Block` to the following command `\@@_NullBlock:` which has the same syntax as the standard command `\Block` but which is no-op.

```
8240  \cs_new_protected:Npn \@@_NullBlock:
8241    { \@@_collect_options:n { \@@_NullBlock_i: } }
8242  \NewExpandableDocumentCommand \@@_NullBlock_i: { m m D < > { } +m }
8243    { }
```

28   How to draw the dotted lines transparently

```
8244  \cs_set_protected:Npn \@@_renew_matrix:
8245    {
8246      \RenewDocumentEnvironment { pmatrix } { }
8247        { \pNiceMatrix }
8248        { \endpNiceMatrix }
8249      \RenewDocumentEnvironment { vmatrix } { }
8250        { \vNiceMatrix }
```

```
8251            { \endvNiceMatrix }
8252        \RenewDocumentEnvironment { Vmatrix } { }
8253            { \VNiceMatrix }
8254            { \endVNiceMatrix }
8255        \RenewDocumentEnvironment { bmatrix } { }
8256            { \bNiceMatrix }
8257            { \endbNiceMatrix }
8258        \RenewDocumentEnvironment { Bmatrix } { }
8259            { \BNiceMatrix }
8260            { \endBNiceMatrix }
8261    }
```

29   Automatic arrays

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

```
8262 \keys_define:nn { nicematrix / Auto }
8263    {
8264        columns-type .tl_set:N = \l_@@_columns_type_tl ,
8265        columns-type .value_required:n = true ,
8266        l .meta:n = { columns-type = l } ,
8267        r .meta:n = { columns-type = r } ,
8268        c .meta:n = { columns-type = c } ,
8269        delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8270        delimiters / color .value_required:n = true ,
8271        delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
8272        delimiters / max-width .default:n = true ,
8273        delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
8274        delimiters .value_required:n = true ,
8275        rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
8276        rounded-corners .default:n = 4 pt
8277    }
8278 \NewDocumentCommand \AutoNiceMatrixWithDelims
8279    { m m O { } > { \SplitArgument { 1 } { - } } m O { } m ! O { } }
8280    { \@@_auto_nice_matrix:nnnnnn { #1 } { #2 } #4 { #6 } { #3 , #5 , #7  } }
8281 \cs_new_protected:Npn \@@_auto_nice_matrix:nnnnnn #1 #2 #3 #4 #5 #6
8282    {
```

The group is for the protection of the keys.

```
8283        \group_begin:
8284        \keys_set_known:nnN { nicematrix / Auto } { #6 } \l_tmpa_tl
8285        \use:e
8286            {
8287                \exp_not:N \begin { NiceArrayWithDelims } { #1 } { #2 }
8288                    { * { #4 } { \exp_not:o \l_@@_columns_type_tl } }
8289                    [ \exp_not:o \l_tmpa_tl ]
8290            }
8291        \int_if_zero:nT \l_@@_first_row_int
8292            {
8293                \int_if_zero:nT \l_@@_first_col_int { & }
8294                \prg_replicate:nn { #4 - 1 } { & }
8295                \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
8296            }
8297        \prg_replicate:nn { #3 }
8298            {
8299                \int_if_zero:nT \l_@@_first_col_int { & }
```

We put { } before #6 to avoid a hasty expansion of a potential \arabic{iRow} at the beginning of the row which would result in an incorrect value of that iRow (since iRow is incremented in the first cell of the row of the \halign).

```
8300                \prg_replicate:nn { #4 - 1 } { { } #5 & } #5
8301                \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
```

```
8302          }
8303      \int_compare:nNnT \l_@@_last_row_int > { -2 }
8304        {
8305          \int_if_zero:nT \l_@@_first_col_int { & }
8306          \prg_replicate:nn { #4 - 1 } { & }
8307          \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
8308        }
8309      \end { NiceArrayWithDelims }
8310      \group_end:
8311    }
8312  \cs_set_protected:Npn \@@_define_com:nnn #1 #2 #3
8313    {
8314      \cs_set_protected:cpn { #1 AutoNiceMatrix }
8315        {
8316          \bool_gset_true:N \g_@@_delims_bool
8317          \str_gset:Ne \g_@@_name_env_str { #1 AutoNiceMatrix }
8318          \AutoNiceMatrixWithDelims { #2 } { #3 }
8319        }
8320    }
8321  \@@_define_com:nnn p ( )
8322  \@@_define_com:nnn b [ ]
8323  \@@_define_com:nnn v | |
8324  \@@_define_com:nnn V \| \|
8325  \@@_define_com:nnn B \{ \}
```

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

```
8326  \NewDocumentCommand \AutoNiceMatrix { O { } m O { } m ! O { } }
8327    {
8328      \group_begin:
8329      \bool_gset_false:N \g_@@_delims_bool
8330      \AutoNiceMatrixWithDelims . . { #2 } { #4 } [ #1 , #3 , #5 ]
8331      \group_end:
8332    }
```

30   The redefinition of the command \dotfill

```
8333  \cs_set_eq:NN \@@_old_dotfill \dotfill
8334  \cs_new_protected:Npn \@@_dotfill:
8335    {
```

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}`).

```
8336      \@@_old_dotfill
8337      \tl_gput_right:Nn \g_@@_cell_after_hook_tl \@@_dotfill_i:
8338    }
```

Now, if the box if not empty (unfornately, 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`.

```
8339  \cs_new_protected:Npn \@@_dotfill_i:
8340    { \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 circonstancies.

```
8341  \cs_new_protected:Npn \@@_diagbox:nn #1 #2
8342    {
8343      \tl_gput_right:Ne \g_@@_pre_code_after_tl
8344        {
8345          \@@_actually_diagbox:nnnnnn
8346            { \int_use:N \c@iRow }
8347            { \int_use:N \c@jCol }
8348            { \int_use:N \c@iRow }
8349            { \int_use:N \c@jCol }
```

`\g_@@_row_style_tl` contains several instructions of the form:

    \@@_if_row_less_than:nn { number } { instructions }

The command `\@@_if_row_less:nn` is fully expandable and, thus, the instructions will be inserted in the `\g_@@_pre_code_after_tl` only if `\diagbox` is used in a row which is the scope of that chunck of instructions.

```
8350          { \g_@@_row_style_tl \exp_not:n { #1 } }
8351          { \g_@@_row_style_tl \exp_not:n { #2 } }
8352        }
```

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`.

```
8353      \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
8354        {
8355          { \int_use:N \c@iRow }
8356          { \int_use:N \c@jCol }
8357          { \int_use:N \c@iRow }
8358          { \int_use:N \c@jCol }
```

The last argument is for the name of the block.

```
8359          { }
8360        }
8361    }
```

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.

```
8362  \cs_new_protected:Npn \@@_actually_diagbox:nnnnnn #1 #2 #3 #4 #5 #6
8363    {
8364      \pgfpicture
8365      \pgf@relevantforpicturesizefalse
8366      \pgfrememberpicturepositiononpagetrue
8367      \@@_qpoint:n { row - #1 }
8368      \dim_set_eq:NN \l_tmpa_dim \pgf@y
8369      \@@_qpoint:n { col - #2 }
8370      \dim_set_eq:NN \l_tmpb_dim \pgf@x
8371      \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
8372      \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
8373      \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8374      \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
8375      \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
8376      \pgfpathlineto { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
8377        {
```

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.

```
8378          \CT@arc@
8379          \pgfsetroundcap
8380          \pgfusepathqstroke
8381        }
8382      \pgfset { inner~sep = 1 pt }
8383      \pgfscope
8384      \pgftransformshift { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
```

```
8385      \pgfnode { rectangle } { south~west }
8386        {
8387          \begin { minipage } { 20 cm }
8388          \@@_math_toggle: #5 \@@_math_toggle:
8389          \end { minipage }
8390        }
8391        { }
8392        { }
8393      \endpgfscope
8394      \pgftransformshift { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
8395      \pgfnode { rectangle } { north~east }
8396        {
8397          \begin { minipage } { 20 cm }
8398          \raggedleft
8399          \@@_math_toggle: #6 \@@_math_toggle:
8400          \end { minipage }
8401        }
8402        { }
8403        { }
8404      \endpgfpicture
8405    }
```

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. 83.

In the environments of nicematrix, \CodeAfter will be linked to \@@_CodeAfter:. That macro must *not* be protected since it begins with \omit.

```
8406 \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 \\.

```
8407 \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.

```
8408 \cs_new_protected:Npn \@@_CodeAfter_ii:n #1 \end
8409   {
8410     \tl_gput_right:Nn \g_nicematrix_code_after_tl { #1 }
8411     \@@_CodeAfter_iv:n
8412   }
```

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

```
8413 \cs_new_protected:Npn \@@_CodeAfter_iv:n #1
8414   {
```

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.

```
8415     \str_if_eq:eeTF \@currenvir { #1 }
8416       { \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.

```
8417       {
8418         \tl_gput_right:Nn \g_nicematrix_code_after_tl { \end { #1 } }
8419         \@@_CodeAfter_ii:n
8420       }
8421   }
```

33 The delimiters in the preamble

The command `\@@_delimiter:nnn` will be used to draw delimiters inside the matrix when delimiters are specified in the preamble of the array. It does *not* concern the exterior delimiters added by {NiceArrayWithDelims} (and {pNiceArray}, {pNiceMatrix}, etc.).

A delimiter in the preamble of the array will write an instruction `\@@_delimiter:nnn` in the `\g_@@_pre_code_after_tl` (and also potentially add instructions in the preamble provided to `\array` in order to add space between columns).

The first argument is the type of delimiter ((, [, \{,),] or \}). The second argument is the number of colummn. 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.

```
8422 \cs_new_protected:Npn \@@_delimiter:nnn #1 #2 #3
8423   {
8424     \pgfpicture
8425     \pgfrememberpicturepositiononpagetrue
8426     \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.

```
8427     \@@_qpoint:n { row - 1 }
8428     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
8429     \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
8430     \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).

```
8431     \bool_if:nTF { #3 }
8432       { \dim_set_eq:NN \l_tmpa_dim \c_max_dim }
8433       { \dim_set:Nn \l_tmpa_dim { - \c_max_dim } }
8434     \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
8435       {
8436         \cs_if_exist:cT
8437           { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
8438           {
8439             \pgfpointanchor
8440               { \@@_env: - ##1 - #2 }
8441               { \bool_if:nTF { #3 } { west } { east } }
8442             \dim_set:Nn \l_tmpa_dim
8443               { \bool_if:nTF { #3 } \dim_min:nn \dim_max:nn \l_tmpa_dim \pgf@x }
8444           }
8445       }
```

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

```
8446     \pgfset { inner~sep = \c_zero_dim }
8447     \dim_zero:N \nulldelimiterspace
8448     \pgftransformshift
8449       {
8450         \pgfpoint
8451           { \l_tmpa_dim }
8452           { ( \l_@@_y_initial_dim + \l_@@_y_final_dim + \arrayrulewidth ) / 2 }
8453       }
8454     \pgfnode
8455       { rectangle }
8456       { \bool_if:nTF { #3 } { east } { west } }
8457       {
```

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

```
8458         \nullfont
8459         \c_math_toggle_token
8460         \@@_color:o \l_@@_delimiters_color_tl
8461         \bool_if:nTF { #3 } { \left #1 } { \left . }
```

```
8462          \vcenter
8463            {
8464              \nullfont
8465              \hrule \@height
8466                      \dim_eval:n { \l_@@_y_initial_dim - \l_@@_y_final_dim }
8467                      \@depth \c_zero_dim
8468                      \@width \c_zero_dim
8469            }
8470          \bool_if:nTF { #3 } { \right . } { \right #1 }
8471          \c_math_toggle_token
8472        }
8473        { }
8474        { }
8475      \endpgfpicture
8476    }
```

34   The command \SubMatrix

```
8477 \keys_define:nn { nicematrix / sub-matrix }
8478    {
8479      extra-height .dim_set:N = \l_@@_submatrix_extra_height_dim ,
8480      extra-height .value_required:n = true ,
8481      left-xshift .dim_set:N = \l_@@_submatrix_left_xshift_dim ,
8482      left-xshift .value_required:n = true ,
8483      right-xshift .dim_set:N = \l_@@_submatrix_right_xshift_dim ,
8484      right-xshift .value_required:n = true ,
8485      xshift .meta:n = { left-xshift = #1, right-xshift = #1 } ,
8486      xshift .value_required:n = true ,
8487      delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8488      delimiters / color .value_required:n = true ,
8489      slim .bool_set:N = \l_@@_submatrix_slim_bool ,
8490      slim .default:n = true ,
8491      hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
8492      hlines .default:n = all ,
8493      vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
8494      vlines .default:n = all ,
8495      hvlines .meta:n = { hlines, vlines } ,
8496      hvlines .value_forbidden:n = true
8497    }
8498 \keys_define:nn { nicematrix }
8499    {
8500      SubMatrix .inherit:n = nicematrix / sub-matrix ,
8501      NiceArray / sub-matrix .inherit:n = nicematrix / sub-matrix ,
8502      pNiceArray / sub-matrix .inherit:n = nicematrix / sub-matrix ,
8503      NiceMatrixOptions / sub-matrix .inherit:n = nicematrix / sub-matrix ,
8504    }
```

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

```
8505 \keys_define:nn { nicematrix / SubMatrix }
8506    {
8507      delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8508      delimiters / color .value_required:n = true ,
8509      hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
8510      hlines .default:n = all ,
8511      vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
8512      vlines .default:n = all ,
8513      hvlines .meta:n = { hlines, vlines } ,
8514      hvlines .value_forbidden:n = true ,
8515      name .code:n =
```

```
8516        \tl_if_empty:nTF { #1 }
8517          { \@@_error:n { Invalid~name } }
8518          {
8519            \regex_match:nnTF { \A[A-Za-z][A-Za-z0-9]*\Z } { #1 }
8520              {
8521                \seq_if_in:NnTF \g_@@_submatrix_names_seq { #1 }
8522                  { \@@_error:nn { Duplicate~name~for~SubMatrix } { #1 } }
8523                  {
8524                    \str_set:Nn \l_@@_submatrix_name_str { #1 }
8525                    \seq_gput_right:Nn \g_@@_submatrix_names_seq { #1 }
8526                  }
8527              }
8528              { \@@_error:n { Invalid~name } }
8529          } ,
8530    name .value_required:n = true ,
8531    rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
8532    rules .value_required:n = true ,
8533    code .tl_set:N = \l_@@_code_tl ,
8534    code .value_required:n = true ,
8535    unknown .code:n = \@@_error:n { Unknown~key~for~SubMatrix }
8536  }


8537 \NewDocumentCommand \@@_SubMatrix_in_code_before { m m m m ! O { } }
8538   {
8539     \peek_remove_spaces:n
8540       {
8541         \tl_gput_right:Ne \g_@@_pre_code_after_tl
8542           {
8543             \SubMatrix { #1 } { #2 } { #3 } { #4 }
8544               [
8545                 delimiters / color = \l_@@_delimiters_color_tl ,
8546                 hlines = \l_@@_submatrix_hlines_clist ,
8547                 vlines = \l_@@_submatrix_vlines_clist ,
8548                 extra-height = \dim_use:N \l_@@_submatrix_extra_height_dim ,
8549                 left-xshift = \dim_use:N \l_@@_submatrix_left_xshift_dim ,
8550                 right-xshift = \dim_use:N \l_@@_submatrix_right_xshift_dim ,
8551                 slim = \bool_to_str:N \l_@@_submatrix_slim_bool ,
8552                 #5
8553               ]
8554           }
8555         \@@_SubMatrix_in_code_before_i { #2 } { #3 }
8556       }
8557  }

8558 \NewDocumentCommand \@@_SubMatrix_in_code_before_i
8559   { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
8560   { \@@_SubMatrix_in_code_before_i:nnnn #1 #2 }
8561 \cs_new_protected:Npn \@@_SubMatrix_in_code_before_i:nnnn #1 #2 #3 #4
8562   {
8563     \seq_gput_right:Ne \g_@@_submatrix_seq
8564       {
```

We use `\str_if_eq:nnTF` because it is fully expandable.

```
8565         { \str_if_eq:nnTF { #1 } { last } { \int_use:N \c@iRow } { #1 } }
8566         { \str_if_eq:nnTF { #2 } { last } { \int_use:N \c@jCol } { #2 } }
8567         { \str_if_eq:nnTF { #3 } { last } { \int_use:N \c@iRow } { #3 } }
8568         { \str_if_eq:nnTF { #4 } { last } { \int_use:N \c@jCol } { #4 } }
8569       }
8570  }
```

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`.

```
8571  \hook_gput_code:nnn { begindocument } { . }
8572    {
8573      \cs_set_nopar:Npn \l_@@_argspec_tl { m m m m O { } E { _ ^ } { { } { } { } } }
8574      \tl_set_rescan:Nno  \l_@@_argspec_tl { } \l_@@_argspec_tl
8575      \exp_args:NNo \NewDocumentCommand \@@_SubMatrix \l_@@_argspec_tl
8576        {
8577          \peek_remove_spaces:n
8578            {
8579              \@@_sub_matrix:nnnnnnn
8580                { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } { #7 }
8581            }
8582        }
8583    }
```

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`).

```
8584  \NewDocumentCommand \@@_compute_i_j:nn
8585    { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
8586    { \@@_compute_i_j:nnnn #1 #2 }
8587  \cs_new_protected:Npn \@@_compute_i_j:nnnn #1 #2 #3 #4
8588    {
8589      \cs_set_nopar:Npn \l_@@_first_i_tl { #1 }
8590      \cs_set_nopar:Npn \l_@@_first_j_tl { #2 }
8591      \cs_set_nopar:Npn \l_@@_last_i_tl { #3 }
8592      \cs_set_nopar:Npn \l_@@_last_j_tl { #4 }
8593      \tl_if_eq:NnT \l_@@_first_i_tl { last }
8594        { \tl_set:NV \l_@@_first_i_tl \c@iRow }
8595      \tl_if_eq:NnT \l_@@_first_j_tl { last }
8596        { \tl_set:NV \l_@@_first_j_tl \c@jCol }
8597      \tl_if_eq:NnT \l_@@_last_i_tl { last }
8598        { \tl_set:NV \l_@@_last_i_tl \c@iRow }
8599      \tl_if_eq:NnT \l_@@_last_j_tl { last }
8600        { \tl_set:NV \l_@@_last_j_tl \c@jCol }
8601    }
8602  \cs_new_protected:Npn \@@_sub_matrix:nnnnnnn #1 #2 #3 #4 #5 #6 #7
8603    {
8604      \group_begin:
```

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

```
8605      \@@_compute_i_j:nn { #2 } { #3 }
8606      \int_compare:nNnT \l_@@_first_i_tl = \l_@@_last_i_tl
8607        { \cs_set_nopar:Npn \arraystretch { 1 } }
8608      \bool_lazy_or:nnTF
8609        { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
8610        { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
8611        { \@@_error:nn { Construct~too~large } { \SubMatrix } }
8612        {
8613          \str_clear_new:N \l_@@_submatrix_name_str
8614          \keys_set:nn { nicematrix / SubMatrix } { #5 }
```

```
8615        \pgfpicture
8616        \pgfrememberpicturepositiononpagetrue
8617        \pgf@relevantforpicturesizefalse
8618        \pgfset { inner~sep = \c_zero_dim }
8619        \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
8620        \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
```

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

```
8621        \bool_if:NTF \l_@@_submatrix_slim_bool
8622          { \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl }
8623          { \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int }
8624          {
8625            \cs_if_exist:cT
8626              { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
8627              {
8628                \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
8629                \dim_set:Nn \l_@@_x_initial_dim
8630                  { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
8631              }
8632            \cs_if_exist:cT
8633              { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
8634              {
8635                \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
8636                \dim_set:Nn \l_@@_x_final_dim
8637                  { \dim_max:nn \l_@@_x_final_dim \pgf@x }
8638              }
8639          }
8640        \dim_compare:nNnTF \l_@@_x_initial_dim = \c_max_dim
8641          { \@@_error:nn { Impossible~delimiter } { left } }
8642          {
8643            \dim_compare:nNnTF \l_@@_x_final_dim = { - \c_max_dim }
8644              { \@@_error:nn { Impossible~delimiter } { right } }
8645              { \@@_sub_matrix_i:nnnn { #1 } { #4 } { #6 } { #7 } }
8646          }
8647        \endpgfpicture
8648      }
8649    \group_end:
8650  }
```

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

```
8651 \cs_new_protected:Npn \@@_sub_matrix_i:nnnn #1 #2 #3 #4
8652   {
8653     \@@_qpoint:n { row - \l_@@_first_i_tl - base }
8654     \dim_set:Nn \l_@@_y_initial_dim
8655       {
8656         \fp_to_dim:n
8657           {
8658             \pgf@y
8659             + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
8660           }
8661       }
8662     \@@_qpoint:n { row - \l_@@_last_i_tl - base }
8663     \dim_set:Nn \l_@@_y_final_dim
8664       { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
8665     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
8666       {
8667         \cs_if_exist:cT
8668           { pgf @ sh @ ns @ \@@_env: - \l_@@_first_i_tl - ##1 }
8669           {
8670             \pgfpointanchor { \@@_env: - \l_@@_first_i_tl - ##1 } { north }
8671             \dim_set:Nn \l_@@_y_initial_dim
8672               { \dim_max:nn \l_@@_y_initial_dim \pgf@y }
8673           }
```

```
8674          \cs_if_exist:cT
8675            { pgf @ sh @ ns @ \@@_env: - \l_@@_last_i_tl - ##1 }
8676            {
8677              \pgfpointanchor { \@@_env: - \l_@@_last_i_tl - ##1 } { south }
8678              \dim_set:Nn \l_@@_y_final_dim
8679                { \dim_min:nn \l_@@_y_final_dim \pgf@y }
8680            }
8681        }
8682      \dim_set:Nn \l_tmpa_dim
8683        {
8684          \l_@@_y_initial_dim - \l_@@_y_final_dim +
8685          \l_@@_submatrix_extra_height_dim - \arrayrulewidth
8686        }
8687      \dim_zero:N \nulldelimiterspace
```

We will draw the rules in the \SubMatrix.

```
8688        \group_begin:
8689        \pgfsetlinewidth { 1.1 \arrayrulewidth }
8690        \@@_set_CT@arc@:o \l_@@_rules_color_tl
8691        \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.

```
8692        \seq_map_inline:Nn \g_@@_cols_vlism_seq
8693          {
8694            \int_compare:nNnT \l_@@_first_j_tl < { ##1 }
8695              {
8696                \int_compare:nNnT
8697                  { ##1 } < { \int_eval:n { \l_@@_last_j_tl + 1 } }
8698                  {
```

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

```
8699                    \@@_qpoint:n { col - ##1 }
8700                    \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
8701                    \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
8702                    \pgfusepathqstroke
8703                  }
8704              }
8705          }
```

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.

```
8706        \tl_if_eq:NNTF \l_@@_submatrix_vlines_clist \c_@@_all_tl
8707          { \int_step_inline:nn { \l_@@_last_j_tl - \l_@@_first_j_tl } }
8708          { \clist_map_inline:Nn \l_@@_submatrix_vlines_clist }
8709          {
8710            \bool_lazy_and:nnTF
8711              { \int_compare_p:nNn { ##1 } > \c_zero_int }
8712              {
8713                \int_compare_p:nNn
8714                  { ##1 } < { \l_@@_last_j_tl - \l_@@_first_j_tl + 1 } }
8715              {
8716                \@@_qpoint:n { col - \int_eval:n { ##1 + \l_@@_first_j_tl } }
8717                \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
8718                \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
8719                \pgfusepathqstroke
8720              }
8721              { \@@_error:nnn { Wrong~line~in~SubMatrix } { vertical } { ##1 } }
8722          }
```

Now, we draw the horizontal rules specified in the key hlines of \SubMatrix. The last argument of \int_step_inline:nn or \clist_map_inline:Nn is given by curryfication.

```
8723        \tl_if_eq:NNTF \l_@@_submatrix_hlines_clist \c_@@_all_tl
8724          { \int_step_inline:nn { \l_@@_last_i_tl - \l_@@_first_i_tl } }
8725          { \clist_map_inline:Nn \l_@@_submatrix_hlines_clist }
8726          {
8727            \bool_lazy_and:nnTF
8728              { \int_compare_p:nNn { ##1 } > \c_zero_int }
8729              {
8730                \int_compare_p:nNn
8731                  { ##1 } < { \l_@@_last_i_tl - \l_@@_first_i_tl + 1 } }
8732              {
8733                \@@_qpoint:n { row - \int_eval:n { ##1 + \l_@@_first_i_tl } }
```

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

```
8734                \group_begin:
```

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

```
8735                \dim_set:Nn \l_tmpa_dim
8736                  { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
8737                \str_case:nn { #1 }
8738                  {
8739                    (  { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
8740                    [  { \dim_sub:Nn \l_tmpa_dim { 0.2 mm } }
8741                    \{ { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
8742                  }
8743                \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }
```

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

```
8744                \dim_set:Nn \l_tmpb_dim
8745                  { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
8746                \str_case:nn { #2 }
8747                  {
8748                    )  { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
8749                    ]  { \dim_add:Nn \l_tmpb_dim { 0.2 mm } }
8750                    \} { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
8751                  }
8752                \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
8753                \pgfusepathqstroke
8754                \group_end:
8755              }
8756              { \@@_error:nnn { Wrong~line~in~SubMatrix } { horizontal } { ##1 } }
8757          }
```

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).

```
8758        \str_if_empty:NF \l_@@_submatrix_name_str
8759          {
8760            \@@_pgf_rect_node:nnnnn \l_@@_submatrix_name_str
8761              \l_@@_x_initial_dim \l_@@_y_initial_dim
8762              \l_@@_x_final_dim \l_@@_y_final_dim
8763          }
8764        \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.

```
8765        \begin { pgfscope }
8766        \pgftransformshift
8767          {
8768            \pgfpoint
8769              { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
8770              { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
8771          }
8772        \str_if_empty:NTF \l_@@_submatrix_name_str
8773          { \@@_node_left:nn #1 { } }
```

```
8774        { \@@_node_left:nn #1 { \@@_env: - \l_@@_submatrix_name_str - left } }
8775      \end { pgfscope }
```

Now, we deal with the right delimiter.

```
8776      \pgftransformshift
8777        {
8778          \pgfpoint
8779            { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
8780            { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
8781        }
8782      \str_if_empty:NTF \l_@@_submatrix_name_str
8783        { \@@_node_right:nnnn #2 { } { #3 } { #4 } }
8784        {
8785          \@@_node_right:nnnn #2
8786            { \@@_env: - \l_@@_submatrix_name_str - right } { #3 } { #4 }
8787        }
8788      \cs_set_eq:NN \pgfpointanchor \@@_pgfpointanchor:n
8789      \flag_clear_new:N \l_@@_code_flag
8790      \l_@@_code_tl
8791    }
```

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

```
8792 \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.

```
8793 \cs_new_protected:Npn \@@_pgfpointanchor:n #1
8794   {
8795     \use:e
8796       { \exp_not:N \@@_old_pgfpointanchor { \@@_pgfpointanchor_i:nn #1 } }
8797   }
```

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 `-`.

```
8798 \cs_new:Npn \@@_pgfpointanchor_i:nn #1 #2
8799   { #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.

```
8800 \tl_const:Nn \c_@@_integers_alist_tl
8801   {
8802     { 1 } { } { 2 } { } { 3 } { } { 4 } { } { 5 } { }
8803     { 6 } { } { 7 } { } { 8 } { } { 9 } { } { 10 } { }
8804     { 11 } { } { 12 } { } { 13 } { } { 14 } { } { 15 } { }
8805     { 16 } { } { 17 } { } { 18 } { } { 19 } { } { 20 } { }
8806   }
```

```
8807 \cs_new:Npn \@@_pgfpointanchor_ii:w #1-#2\q_stop
8808   {
```

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`.

```
8809        \tl_if_empty:nTF { #2 }
8810          {
8811            \str_case:nVTF { #1 } \c_@@_integers_alist_tl
8812              {
8813                \flag_raise:N \l_@@_code_flag
8814                \int_if_even:nTF { \flag_height:N \l_@@_code_flag }
8815                  { \int_eval:n { #1 + \l_@@_first_i_tl - 1 } }
8816                  { \int_eval:n { #1 + \l_@@_first_j_tl - 1 } }
8817              }
8818              { #1 }
8819          }
```

If there is an hyphen, we have to see whether we have a node of the form i-j, `row`-i or `col`-j.

```
8820          { \@@_pgfpointanchor_iii:w { #1 } #2 }
8821      }
```

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`).

```
8822  \cs_new:Npn \@@_pgfpointanchor_iii:w #1 #2 -
8823    {
8824      \str_case:nnF { #1 }
8825        {
8826          { row } { row - \int_eval:n { #2 + \l_@@_first_i_tl - 1 } }
8827          { col } { col - \int_eval:n { #2 + \l_@@_first_j_tl - 1 } }
8828        }
```

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

```
8829        {
8830          \int_eval:n { #1 + \l_@@_first_i_tl - 1 }
8831          - \int_eval:n { #2 + \l_@@_first_j_tl - 1 }
8832        }
8833    }
```

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`).

```
8834  \cs_new_protected:Npn \@@_node_left:nn #1 #2
8835    {
8836      \pgfnode
8837        { rectangle }
8838        { east }
8839        {
8840          \nullfont
8841          \c_math_toggle_token
8842          \@@_color:o \l_@@_delimiters_color_tl
8843          \left #1
8844          \vcenter
8845            {
8846              \nullfont
8847              \hrule \@height \l_tmpa_dim
8848                     \@depth \c_zero_dim
8849                     \@width \c_zero_dim
8850            }
8851          \right .
8852          \c_math_toggle_token
8853        }
8854        { #2 }
```

```
8855        { }
8856    }
```

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.

```
8857  \cs_new_protected:Npn \@@_node_right:nnnn #1 #2 #3 #4
8858    {
8859      \pgfnode
8860        { rectangle }
8861        { west }
8862        {
8863          \nullfont
8864          \c_math_toggle_token
8865          \colorlet { current-color } { . }
8866          \@@_color:o \l_@@_delimiters_color_tl
8867          \left .
8868          \vcenter
8869            {
8870              \nullfont
8871              \hrule \@height \l_tmpa_dim
8872                     \@depth \c_zero_dim
8873                     \@width \c_zero_dim
8874            }
8875          \right #1
8876          \tl_if_empty:nF { #3 } { _ { \smash { #3 } } }
8877          ^ { \color { current-color } \smash { #4 } }
8878          \c_math_toggle_token
8879        }
8880        { #2 }
8881        { }
8882    }
```

35   Les commandes \UnderBrace et \OverBrace

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

```
8883  \NewDocumentCommand \@@_UnderBrace { O { } m m m O { } }
8884    {
8885      \peek_remove_spaces:n
8886        { \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { under } }
8887    }
8888  \NewDocumentCommand \@@_OverBrace { O { } m m m O { } }
8889    {
8890      \peek_remove_spaces:n
8891        { \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { over } }
8892    }
8893  \keys_define:nn { nicematrix / Brace }
8894    {
8895      left-shorten .bool_set:N = \l_@@_brace_left_shorten_bool ,
8896      left-shorten .default:n = true ,
8897      left-shorten .value_forbidden:n = true ,
8898      right-shorten .bool_set:N = \l_@@_brace_right_shorten_bool ,
8899      right-shorten .default:n = true ,
8900      right-shorten .value_forbidden:n = true ,
8901      shorten .meta:n = { left-shorten , right-shorten } ,
8902      shorten .value_forbidden:n = true ,
8903      yshift .dim_set:N = \l_@@_brace_yshift_dim ,
```

```
8904      yshift .value_required:n = true ,
8905      yshift .initial:n = \c_zero_dim ,
8906      color .tl_set:N = \l_tmpa_tl ,
8907      color .value_required:n = true ,
8908      unknown .code:n = \@@_error:n { Unknown~key~for~Brace }
8909    }
```

#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`.

```
8910  \cs_new_protected:Npn \@@_brace:nnnnn #1 #2 #3 #4 #5
8911    {
8912      \group_begin:
```

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

```
8913      \@@_compute_i_j:nn { #1 } { #2 }
8914      \bool_lazy_or:nnTF
8915        { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
8916        { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
8917        {
8918          \str_if_eq:nnTF { #5 } { under }
8919            { \@@_error:nn { Construct~too~large } { \UnderBrace } }
8920            { \@@_error:nn { Construct~too~large } { \OverBrace } }
8921        }
8922        {
8923          \tl_clear:N \l_tmpa_tl
8924          \keys_set:nn { nicematrix / Brace } { #4 }
8925          \tl_if_empty:NF \l_tmpa_tl { \color { \l_tmpa_tl } }
8926          \pgfpicture
8927          \pgfrememberpicturepositiononpagetrue
8928          \pgf@relevantforpicturesizefalse
8929          \bool_if:NT \l_@@_brace_left_shorten_bool
8930            {
8931              \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
8932              \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
8933                {
8934                  \cs_if_exist:cT
8935                    { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
8936                    {
8937                      \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
8938                      \dim_set:Nn \l_@@_x_initial_dim
8939                        { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
8940                    }
8941                }
8942            }
8943          \bool_lazy_or:nnT
8944            { \bool_not_p:n \l_@@_brace_left_shorten_bool }
8945            { \dim_compare_p:nNn \l_@@_x_initial_dim = \c_max_dim }
8946            {
8947              \@@_qpoint:n { col - \l_@@_first_j_tl }
8948              \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
8949            }
8950          \bool_if:NT \l_@@_brace_right_shorten_bool
8951            {
8952              \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
8953              \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
8954                {
8955                  \cs_if_exist:cT
8956                    { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
8957                    {
8958                      \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
8959                      \dim_set:Nn \l_@@_x_final_dim
8960                        { \dim_max:nn \l_@@_x_final_dim \pgf@x }
8961                    }
```

```
8962                           }
8963                        }
8964                 \bool_lazy_or:nnT
8965                   { \bool_not_p:n \l_@@_brace_right_shorten_bool }
8966                   { \dim_compare_p:nNn \l_@@_x_final_dim = { - \c_max_dim } }
8967                   {
8968                      \@@_qpoint:n { col - \int_eval:n { \l_@@_last_j_tl + 1 } }
8969                      \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
8970                   }
8971                 \pgfset { inner~sep = \c_zero_dim }
8972                 \str_if_eq:nnTF { #5 } { under }
8973                   { \@@_underbrace_i:n { #3 } }
8974                   { \@@_overbrace_i:n { #3 } }
8975                 \endpgfpicture
8976             }
8977       \group_end:
8978    }
```

The argument is the text to put above the brace.

```
8979 \cs_new_protected:Npn \@@_overbrace_i:n #1
8980    {
8981       \@@_qpoint:n { row - \l_@@_first_i_tl }
8982       \pgftransformshift
8983          {
8984             \pgfpoint
8985               { ( \l_@@_x_initial_dim + \l_@@_x_final_dim) / 2 }
8986               { \pgf@y + \l_@@_brace_yshift_dim - 3 pt}
8987          }
8988       \pgfnode
8989          { rectangle }
8990          { south }
8991          {
8992             \vtop
8993               {
8994                  \group_begin:
8995                  \everycr { }
8996                  \halign
8997                    {
8998                       \hfil ## \hfil \crcr
8999                       \@@_math_toggle: #1 \@@_math_toggle: \cr
9000                       \noalign { \skip_vertical:n { 3 pt } \nointerlineskip }
9001                       \c_math_toggle_token
9002                       \overbrace
9003                         {
9004                            \hbox_to_wd:nn
9005                              { \l_@@_x_final_dim - \l_@@_x_initial_dim }
9006                              { }
9007                         }
9008                       \c_math_toggle_token
9009                       \cr
9010                    }
9011                  \group_end:
9012               }
9013          }
9014          { }
9015          { }
9016    }
```

The argument is the text to put under the brace.

```
9017 \cs_new_protected:Npn \@@_underbrace_i:n #1
9018    {
9019       \@@_qpoint:n { row - \int_eval:n { \l_@@_last_i_tl + 1 } }
9020       \pgftransformshift
```

```
9021        {
9022          \pgfpoint
9023            { ( \l_@@_x_initial_dim + \l_@@_x_final_dim) / 2 }
9024            { \pgf@y  - \l_@@_brace_yshift_dim + 3 pt }
9025        }
9026      \pgfnode
9027        { rectangle }
9028        { north }
9029        {
9030          \group_begin:
9031          \everycr { }
9032          \vbox
9033            {
9034              \halign
9035                {
9036                  \hfil ## \hfil \crcr
9037                  \c_math_toggle_token
9038                  \underbrace
9039                    {
9040                      \hbox_to_wd:nn
9041                        { \l_@@_x_final_dim - \l_@@_x_initial_dim }
9042                        { }
9043                    }
9044                  \c_math_toggle_token
9045                  \cr
9046                  \noalign { \skip_vertical:n { 3 pt } \nointerlineskip }
9047                  \@@_math_toggle: #1 \@@_math_toggle: \cr
9048                }
9049            }
9050          \group_end:
9051        }
9052        { }
9053        { }
9054    }
```

36   The command TikzEveryCell

```
9055 \bool_new:N \l_@@_not_empty_bool
9056 \bool_new:N \l_@@_empty_bool
9057
9058 \keys_define:nn { nicematrix / TikzEveryCell }
9059   {
9060     not-empty .code:n =
9061       \bool_lazy_or:nnTF
9062         \l_@@_in_code_after_bool
9063         \g_@@_recreate_cell_nodes_bool
9064         { \bool_set_true:N \l_@@_not_empty_bool }
9065         { \@@_error:n { detection~of~empty~cells } } ,
9066     not-empty .value_forbidden:n = true ,
9067     empty .code:n =
9068       \bool_lazy_or:nnTF
9069         \l_@@_in_code_after_bool
9070         \g_@@_recreate_cell_nodes_bool
9071         { \bool_set_true:N \l_@@_empty_bool }
9072         { \@@_error:n { detection~of~empty~cells } } ,
9073     empty .value_forbidden:n = true ,
9074     unknown .code:n = \@@_error:n { Unknown~key~for~TikzEveryCell }
9075   }
9076
```

```
9077
9078  \NewDocumentCommand { \@@_TikzEveryCell } { O { } m }
9079    {
9080      \IfPackageLoadedTF { tikz }
9081        {
9082          \group_begin:
9083          \keys_set:nn { nicematrix / TikzEveryCell } { #1 }
```

The inner pair of braces in the following line is mandatory because, the last argument of
\@@_tikz:nnnnn is *a list of lists* of TikZ keys.

```
9084          \tl_set:Nn \l_tmpa_tl { { #2 } }
9085          \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
9086            { \@@_for_a_block:nnnnn ##1 }
9087          \@@_all_the_cells:
9088          \group_end:
9089        }
9090        { \@@_error:n { TikzEveryCell~without~tikz } }
9091    }
9092
9093  \tl_new:N \@@_i_tl
9094  \tl_new:N \@@_j_tl
9095
9096  \cs_new_protected:Nn \@@_all_the_cells:
9097    {
9098      \int_step_variable:nNn { \int_use:c { c@iRow } } \@@_i_tl
9099        {
9100          \int_step_variable:nNn { \int_use:c { c@jCol } } \@@_j_tl
9101            {
9102              \cs_if_exist:cF { cell - \@@_i_tl - \@@_j_tl }
9103                {
9104                  \exp_args:NNe \seq_if_in:NnF \l_@@_corners_cells_seq
9105                    { \@@_i_tl - \@@_j_tl }
9106                    {
9107                      \bool_set_false:N \l_tmpa_bool
9108                      \cs_if_exist:cTF
9109                        { pgf @ sh @ ns @ \@@_env: - \@@_i_tl - \@@_j_tl }
9110                        {
9111                          \bool_if:NF \l_@@_empty_bool
9112                            { \bool_set_true:N \l_tmpa_bool }
9113                        }
9114                        {
9115                          \bool_if:NF \l_@@_not_empty_bool
9116                            { \bool_set_true:N \l_tmpa_bool }
9117                        }
9118                      \bool_if:NT \l_tmpa_bool
9119                        {
9120                          \@@_block_tikz:nnnnV
9121                          \@@_i_tl \@@_j_tl \@@_i_tl \@@_j_tl \l_tmpa_tl
9122                        }
9123                    }
9124                }
9125            }
9126        }
9127    }
9128
9129  \cs_new_protected:Nn \@@_for_a_block:nnnnn
9130    {
9131      \bool_if:NF \l_@@_empty_bool
9132        {
9133          \@@_block_tikz:nnnnV
9134            { #1 } { #2 } { #3 } { #4 } \l_tmpa_tl
9135        }
9136      \@@_mark_cells_of_block:nnnn { #1 } { #2 } { #3 } { #4 }
9137    }
```

206

```
9138
9139  \cs_new_protected:Nn \@@_mark_cells_of_block:nnnn
9140    {
9141      \int_step_inline:nnn { #1 } { #3 }
9142        {
9143          \int_step_inline:nnn { #2 } { #4 }
9144            { \cs_set:cpn { cell - ##1 - ####1 } { } } }
9145        }
9146    }
```

37   The command \ShowCellNames

```
9147  \NewDocumentCommand \@@_ShowCellNames_CodeBefore { }
9148    {
9149      \dim_gzero_new:N \g_@@_tmpc_dim
9150      \dim_gzero_new:N \g_@@_tmpd_dim
9151      \dim_gzero_new:N \g_@@_tmpe_dim
9152      \int_step_inline:nn \c@iRow
9153        {
9154          \begin { pgfpicture }
9155          \@@_qpoint:n { row - ##1 }
9156          \dim_set_eq:NN \l_tmpa_dim \pgf@y
9157          \@@_qpoint:n { row - \int_eval:n { ##1 + 1 } }
9158          \dim_gset:Nn \g_tmpa_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
9159          \dim_gset:Nn \g_tmpb_dim { \l_tmpa_dim - \pgf@y }
9160          \bool_if:NTF \l_@@_in_code_after_bool
9161          \end { pgfpicture }
9162          \int_step_inline:nn \c@jCol
9163            {
9164              \hbox_set:Nn \l_tmpa_box
9165                { \normalfont \Large \color { red ! 50 } ##1 - ####1 }
9166              \begin { pgfpicture }
9167              \@@_qpoint:n { col - ####1 }
9168              \dim_gset_eq:NN \g_@@_tmpc_dim \pgf@x
9169              \@@_qpoint:n { col - \int_eval:n { ####1 + 1 } }
9170              \dim_gset:Nn \g_@@_tmpd_dim { \pgf@x - \g_@@_tmpc_dim }
9171              \dim_gset_eq:NN \g_@@_tmpe_dim \pgf@x
9172              \endpgfpicture
9173              \end { pgfpicture }
9174              \fp_set:Nn \l_tmpa_fp
9175                {
9176                  \fp_min:nn
9177                    {
9178                      \fp_min:nn
9179                        {
9180                          \dim_ratio:nn
9181                            { \g_@@_tmpd_dim }
9182                            { \box_wd:N \l_tmpa_box }
9183                        }
9184                        {
9185                          \dim_ratio:nn
9186                            { \g_tmpb_dim }
9187                            { \box_ht_plus_dp:N \l_tmpa_box }
9188                        }
9189                    }
9190                    { 1.0 }
9191                }
9192              \box_scale:Nnn \l_tmpa_box
9193                { \fp_use:N \l_tmpa_fp }
9194                { \fp_use:N \l_tmpa_fp }
9195              \pgfpicture
9196              \pgfrememberpicturepositiononpagetrue
```

```
9197            \pgf@relevantforpicturesizefalse
9198            \pgftransformshift
9199              {
9200                \pgfpoint
9201                  { 0.5 * ( \g_@@_tmpc_dim + \g_@@_tmpe_dim ) }
9202                  { \dim_use:N \g_tmpa_dim }
9203              }
9204            \pgfnode
9205              { rectangle }
9206              { center }
9207              { \box_use:N \l_tmpa_box }
9208              { }
9209              { }
9210            \endpgfpicture
9211          }
9212      }
9213  }
9214  \NewDocumentCommand \@@_ShowCellNames { }
9215    {
9216      \bool_if:NT \l_@@_in_code_after_bool
9217        {
9218          \pgfpicture
9219          \pgfremberpicturepositiononpagetrue
9220          \pgf@relevantforpicturesizefalse
9221          \pgfpathrectanglecorners
9222            { \@@_qpoint:n { 1 } }
9223            {
9224              \@@_qpoint:n
9225                { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
9226            }
9227          \pgfsetfillopacity { 0.75 }
9228          \pgfsetfillcolor { white }
9229          \pgfusepathqfill
9230          \endpgfpicture
9231        }
9232      \dim_gzero_new:N \g_@@_tmpc_dim
9233      \dim_gzero_new:N \g_@@_tmpd_dim
9234      \dim_gzero_new:N \g_@@_tmpe_dim
9235      \int_step_inline:nn \c@iRow
9236        {
9237          \bool_if:NTF \l_@@_in_code_after_bool
9238            {
9239              \pgfpicture
9240              \pgfremberpicturepositiononpagetrue
9241              \pgf@relevantforpicturesizefalse
9242            }
9243            { \begin { pgfpicture } }
9244          \@@_qpoint:n { row - ##1 }
9245          \dim_set_eq:NN \l_tmpa_dim \pgf@y
9246          \@@_qpoint:n { row - \int_eval:n { ##1 + 1 } }
9247          \dim_gset:Nn \g_tmpa_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
9248          \dim_gset:Nn \g_tmpb_dim { \l_tmpa_dim - \pgf@y }
9249          \bool_if:NTF \l_@@_in_code_after_bool
9250            { \endpgfpicture }
9251            { \end { pgfpicture } }
9252          \int_step_inline:nn \c@jCol
9253            {
9254              \hbox_set:Nn \l_tmpa_box
9255                {
9256                  \normalfont \Large \sffamily \bfseries
9257                  \bool_if:NTF \l_@@_in_code_after_bool
9258                    { \color { red } }
9259                    { \color { red ! 50 } }
```

208

```
9260                    ##1 - ####1
9261                  }
9262              \bool_if:NTF \l_@@_in_code_after_bool
9263                {
9264                  \pgfpicture
9265                  \pgfrememberpicturepositiononpagetrue
9266                  \pgf@relevantforpicturesizefalse
9267                }
9268                { \begin { pgfpicture } }
9269              \@@_qpoint:n { col - ####1 }
9270              \dim_gset_eq:NN \g_@@_tmpc_dim \pgf@x
9271              \@@_qpoint:n { col - \int_eval:n { ####1 + 1 } }
9272              \dim_gset:Nn \g_@@_tmpd_dim { \pgf@x - \g_@@_tmpc_dim }
9273              \dim_gset_eq:NN \g_@@_tmpe_dim \pgf@x
9274              \bool_if:NTF \l_@@_in_code_after_bool
9275                { \endpgfpicture }
9276                { \end { pgfpicture } }
9277              \fp_set:Nn \l_tmpa_fp
9278                {
9279                  \fp_min:nn
9280                    {
9281                      \fp_min:nn
9282                        { \dim_ratio:nn \g_@@_tmpd_dim { \box_wd:N \l_tmpa_box } }
9283                        { \dim_ratio:nn \g_tmpb_dim { \box_ht_plus_dp:N \l_tmpa_box } }
9284                    }
9285                    { 1.0 }
9286                }
9287              \box_scale:Nnn \l_tmpa_box { \fp_use:N \l_tmpa_fp } { \fp_use:N \l_tmpa_fp }
9288              \pgfpicture
9289              \pgfrememberpicturepositiononpagetrue
9290              \pgf@relevantforpicturesizefalse
9291              \pgftransformshift
9292                {
9293                  \pgfpoint
9294                    { 0.5 * ( \g_@@_tmpc_dim + \g_@@_tmpe_dim ) }
9295                    { \dim_use:N \g_tmpa_dim }
9296                }
9297              \pgfnode
9298                { rectangle }
9299                { center }
9300                { \box_use:N \l_tmpa_box }
9301                { }
9302                { }
9303              \endpgfpicture
9304            }
9305        }
9306  }
```

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.

```
9307  \bool_new:N \g_@@_footnotehyper_bool
```

The boolean `\g_@@_footnote_bool` will indicate if the option footnote is used, but quicky, it will
also be set to true if the option footnotehyper is used.

```
9308  \bool_new:N \g_@@_footnote_bool
```

```
9309  \msg_new:nnnn { nicematrix } { Unknown~key~for~package }
9310    {
9311      The~key~'\l_keys_key_str'~is~unknown. \\
9312      That~key~will~be~ignored. \\
9313      For~a~list~of~the~available~keys,~type~H~<return>.
9314    }
9315    {
9316      The~available~keys~are~(in~alphabetic~order):~
9317      footnote,~
9318      footnotehyper,~
9319      messages-for-Overleaf,~
9320      no-test-for-array,~
9321      renew-dots,~and~
9322      renew-matrix.
9323    }
9324  \keys_define:nn { nicematrix / Package }
9325    {
9326      renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
9327      renew-dots .value_forbidden:n = true ,
9328      renew-matrix .code:n = \@@_renew_matrix: ,
9329      renew-matrix .value_forbidden:n = true ,
9330      messages-for-Overleaf .bool_set:N = \g_@@_messages_for_Overleaf_bool ,
9331      footnote .bool_set:N = \g_@@_footnote_bool ,
9332      footnotehyper .bool_set:N = \g_@@_footnotehyper_bool ,
9333      no-test-for-array .bool_set:N = \g_@@_no_test_for_array_bool ,
9334      no-test-for-array .default:n = true ,
9335      unknown .code:n = \@@_error:n { Unknown~key~for~package }
9336    }
9337  \ProcessKeysOptions { nicematrix / Package }


9338  \@@_msg_new:nn { footnote~with~footnotehyper~package }
9339    {
9340      You~can't~use~the~option~'footnote'~because~the~package~
9341      footnotehyper~has~already~been~loaded.~
9342      If~you~want,~you~can~use~the~option~'footnotehyper'~and~the~footnotes~
9343      within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
9344      of~the~package~footnotehyper.\\
9345      The~package~footnote~won't~be~loaded.
9346    }
9347  \@@_msg_new:nn { footnotehyper~with~footnote~package }
9348    {
9349      You~can't~use~the~option~'footnotehyper'~because~the~package~
9350      footnote~has~already~been~loaded.~
9351      If~you~want,~you~can~use~the~option~'footnote'~and~the~footnotes~
9352      within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
9353      of~the~package~footnote.\\
9354      The~package~footnotehyper~won't~be~loaded.
9355    }


9356  \bool_if:NT \g_@@_footnote_bool
9357    {
```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```
9358      \IfClassLoadedTF { beamer }
9359        { \bool_set_false:N \g_@@_footnote_bool }
9360        {
9361          \IfPackageLoadedTF { footnotehyper }
9362            { \@@_error:n { footnote~with~footnotehyper~package } }
9363            { \usepackage { footnote } }
9364        }
9365    }
```

```
9366  \bool_if:NT \g_@@_footnotehyper_bool
9367    {
```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```
9368      \IfClassLoadedTF { beamer }
9369        { \bool_set_false:N \g_@@_footnote_bool }
9370        {
9371          \IfPackageLoadedTF { footnote }
9372            { \@@_error:n { footnotehyper~with~footnote~package } }
9373            { \usepackage { footnotehyper } }
9374        }
9375      \bool_set_true:N \g_@@_footnote_bool
9376    }
```

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.

```
9377  \bool_new:N \l_@@_underscore_loaded_bool
9378  \IfPackageLoadedTF { underscore }
9379    { \bool_set_true:N \l_@@_underscore_loaded_bool }
9380    { }
9381  \hook_gput_code:nnn { begindocument } { . }
9382    {
9383      \bool_if:NF \l_@@_underscore_loaded_bool
9384        {
9385          \IfPackageLoadedTF { underscore }
9386            { \@@_error:n { underscore~after~nicematrix } }
9387            { }
9388        }
9389    }
```

40   Error messages of the package

```
9390  \bool_if:NTF \g_@@_messages_for_Overleaf_bool
9391    { \str_const:Nn \c_@@_available_keys_str { } }
9392    {
9393      \str_const:Nn \c_@@_available_keys_str
9394        { For~a~list~of~the~available~keys,~type~H~<return>. }
9395    }
9396  \seq_new:N \g_@@_types_of_matrix_seq
9397  \seq_gset_from_clist:Nn \g_@@_types_of_matrix_seq
9398    {
9399      NiceMatrix ,
9400      pNiceMatrix , bNiceMatrix , vNiceMatrix, BNiceMatrix, VNiceMatrix
9401    }
9402  \seq_gset_map_x:NNn \g_@@_types_of_matrix_seq \g_@@_types_of_matrix_seq
9403    { \tl_to_str:n { #1 } }
```

If the user uses too much columns, the command \@@_error_too_much_cols: is triggered. This command raises an error but also tries to give the best information to the user in the error message. The command \seq_if_in:NoTF is not expandable and that's why we can't put it in the error message itself. We have to do the test before the \@@_fatal:n.

```
9404 \cs_new_protected:Npn \@@_error_too_much_cols:
9405   {
9406     \seq_if_in:NoTF \g_@@_types_of_matrix_seq \g_@@_name_env_str
9407       {
9408         \int_compare:nNnTF \l_@@_last_col_int = { -2 }
9409           { \@@_fatal:n { too~much~cols~for~matrix } }
9410           {
9411             \int_compare:nNnTF \l_@@_last_col_int = { -1 }
9412               { \@@_fatal:n { too~much~cols~for~matrix } }
9413               {
9414                 \bool_if:NF \l_@@_last_col_without_value_bool
9415                   { \@@_fatal:n { too~much~cols~for~matrix~with~last~col } }
9416               }
9417           }
9418       }
9419       { \@@_fatal:nn { too~much~cols~for~array } }
9420   }
```

The following command must *not* be protected since it's used in an error message.

```
9421 \cs_new:Npn \@@_message_hdotsfor:
9422   {
9423     \tl_if_empty:oF \g_@@_HVdotsfor_lines_tl
9424       { ~Maybe~your~use~of~\token_to_str:N \Hdotsfor\ is~incorrect.}
9425   }
9426 \@@_msg_new:nn { hvlines,~rounded-corners~and~corners }
9427   {
9428     Incompatible~options.\\
9429     You~should~not~use~'hvlines',~'rounded-corners'~and~'corners'~at~this~time.\\
9430     The~output~will~not~be~reliable.
9431   }
9432 \@@_msg_new:nn { negative~weight }
9433   {
9434     Negative~weight.\\
9435     The~weight~of~the~'X'~columns~must~be~positive~and~you~have~used~
9436     the~value~'\int_use:N \l_@@_weight_int'.\\
9437     The~absolute~value~will~be~used.
9438   }
9439 \@@_msg_new:nn { last~col~not~used }
9440   {
9441     Column~not~used.\\
9442     The~key~'last-col'~is~in~force~but~you~have~not~used~that~last~column~
9443     in~your~\@@_full_name_env:.~However,~you~can~go~on.
9444   }
9445 \@@_msg_new:nn { too~much~cols~for~matrix~with~last~col }
9446   {
9447     Too~much~columns.\\
9448     In~the~row~\int_eval:n { \c@iRow },~
9449     you~try~to~use~more~columns~
9450     than~allowed~by~your~\@@_full_name_env:.\@@_message_hdotsfor:\
9451     The~maximal~number~of~columns~is~\int_eval:n { \l_@@_last_col_int - 1 }~
9452     (plus~the~exterior~columns).~This~error~is~fatal.
9453   }
9454 \@@_msg_new:nn { too~much~cols~for~matrix }
9455   {
9456     Too~much~columns.\\
9457     In~the~row~\int_eval:n { \c@iRow },~
9458     you~try~to~use~more~columns~than~allowed~by~your~
9459     \@@_full_name_env:.\@@_message_hdotsfor:\ Recall~that~the~maximal~
```

```
9460    number~of~columns~for~a~matrix~(excepted~the~potential~exterior~
9461    columns)~is~fixed~by~the~LaTeX~counter~'MaxMatrixCols'.~
9462    Its~current~value~is~\int_use:N \c@MaxMatrixCols\ (use~
9463    \token_to_str:N \setcounter\ to~change~that~value).~
9464    This~error~is~fatal.
9465  }

9466  \@@_msg_new:nn { too~much~cols~for~array }
9467    {
9468    Too~much~columns.\\
9469    In~the~row~\int_eval:n { \c@iRow },~
9470    ~you~try~to~use~more~columns~than~allowed~by~your~
9471    \@@_full_name_env:.\@@_message_hdotsfor:\ The~maximal~number~of~columns~is~
9472    \int_use:N \g_@@_static_num_of_col_int\
9473    ~(plus~the~potential~exterior~ones).~
9474    This~error~is~fatal.
9475  }

9476  \@@_msg_new:nn { columns~not~used }
9477    {
9478    Columns~not~used.\\
9479    The~preamble~of~your~\@@_full_name_env:\ announces~\int_use:N
9480    \g_@@_static_num_of_col_int\ columns~but~you~use~only~\int_use:N \c@jCol.\\
9481    The~columns~you~did~not~used~won't~be~created.\\
9482    You~won't~have~similar~error~message~till~the~end~of~the~document.
9483  }

9484  \@@_msg_new:nn { empty~preamble }
9485    {
9486    Empty~preamble.\\
9487    The~preamble~of~your~\@@_full_name_env:\ is~empty.\\
9488    This~error~is~fatal.
9489  }

9490  \@@_msg_new:nn { in~first~col }
9491    {
9492    Erroneous~use.\\
9493    You~can't~use~the~command~#1 in~the~first~column~(number~0)~of~the~array.\\
9494    That~command~will~be~ignored.
9495  }

9496  \@@_msg_new:nn { in~last~col }
9497    {
9498    Erroneous~use.\\
9499    You~can't~use~the~command~#1 in~the~last~column~(exterior)~of~the~array.\\
9500    That~command~will~be~ignored.
9501  }

9502  \@@_msg_new:nn { in~first~row }
9503    {
9504    Erroneous~use.\\
9505    You~can't~use~the~command~#1 in~the~first~row~(number~0)~of~the~array.\\
9506    That~command~will~be~ignored.
9507  }

9508  \@@_msg_new:nn { in~last~row }
9509    {
9510    You~can't~use~the~command~#1 in~the~last~row~(exterior)~of~the~array.\\
9511    That~command~will~be~ignored.
9512  }

9513  \@@_msg_new:nn { caption~outside~float }
9514    {
9515    Key~caption~forbidden.\\
9516    You~can't~use~the~key~'caption'~because~you~are~not~in~a~floating~
9517    environment.~This~key~will~be~ignored.
9518  }
```

```
9519  \@@_msg_new:nn { short-caption~without~caption }
9520    {
9521      You~should~not~use~the~key~'short-caption'~without~'caption'.~
9522      However,~your~'short-caption'~will~be~used~as~'caption'.
9523    }
9524  \@@_msg_new:nn { double~closing~delimiter }
9525    {
9526      Double~delimiter.\\
9527      You~can't~put~a~second~closing~delimiter~"#1"~just~after~a~first~closing~
9528      delimiter.~This~delimiter~will~be~ignored.
9529    }
9530  \@@_msg_new:nn { delimiter~after~opening }
9531    {
9532      Double~delimiter.\\
9533      You~can't~put~a~second~delimiter~"#1"~just~after~a~first~opening~
9534      delimiter.~That~delimiter~will~be~ignored.
9535    }
9536  \@@_msg_new:nn { bad~option~for~line-style }
9537    {
9538      Bad~line~style.\\
9539      Since~you~haven't~loaded~Tikz,~the~only~value~you~can~give~to~'line-style'~
9540      is~'standard'.~That~key~will~be~ignored.
9541    }
9542  \@@_msg_new:nn { Identical~notes~in~caption }
9543    {
9544      Identical~tabular~notes.\\
9545      You~can't~put~several~notes~with~the~same~content~in~
9546      \token_to_str:N \caption\ (but~you~can~in~the~main~tabular).\\
9547      If~you~go~on,~the~output~will~probably~be~erroneous.
9548    }
9549  \@@_msg_new:nn { tabularnote~below~the~tabular }
9550    {
9551      \token_to_str:N \tabularnote\ forbidden\\
9552      You~can't~use~\token_to_str:N \tabularnote\ in~the~caption~
9553      of~your~tabular~because~the~caption~will~be~composed~below~
9554      the~tabular.~If~you~want~the~caption~above~the~tabular~use~the~
9555      key~'caption-above'~in~\token_to_str:N \NiceMatrixOptions.\\
9556      Your~\token_to_str:N \tabularnote\ will~be~discarded~and~
9557      no~similar~error~will~raised~in~this~document.
9558    }
9559  \@@_msg_new:nn { Unknown~key~for~rules }
9560    {
9561      Unknown~key.\\
9562      There~is~only~two~keys~available~here:~width~and~color.\\
9563      Your~key~'\l_keys_key_str'~will~be~ignored.
9564    }
9565  \@@_msg_new:nn { Unknown~key~for~TikzEveryCell }
9566    {
9567      Unknown~key.\\
9568      There~is~only~two~keys~available~here:~
9569      'empty'~and~'not-empty'.\\
9570      Your~key~'\l_keys_key_str'~will~be~ignored.
9571    }
9572  \@@_msg_new:nn { Unknown~key~for~rotate }
9573    {
9574      Unknown~key.\\
9575      The~only~key~available~here~is~'c'.\\
9576      Your~key~'\l_keys_key_str'~will~be~ignored.
9577    }
9578  \@@_msg_new:nnn { Unknown~key~for~custom-line }
```

```
9579    {
9580      Unknown~key.\\
9581      The~key~'\l_keys_key_str'~is~unknown~in~a~'custom-line'.~
9582      It~you~go~on,~you~will~probably~have~other~errors. \\
9583      \c_@@_available_keys_str
9584    }
9585    {
9586      The~available~keys~are~(in~alphabetic~order):~
9587      ccommand,~
9588      color,~
9589      command,~
9590      dotted,~
9591      letter,~
9592      multiplicity,~
9593      sep-color,~
9594      tikz,~and~total-width.
9595    }
9596  \@@_msg_new:nnn { Unknown~key~for~xdots }
9597    {
9598      Unknown~key.\\
9599      The~key~'\l_keys_key_str'~is~unknown~for~a~command~for~drawing~dotted~rules.\\
9600      \c_@@_available_keys_str
9601    }
9602    {
9603      The~available~keys~are~(in~alphabetic~order):~
9604      'color',~
9605      'horizontal-labels',~
9606      'inter',~
9607      'line-style',~
9608      'radius',~
9609      'shorten',~
9610      'shorten-end'~and~'shorten-start'.
9611    }
9612  \@@_msg_new:nn { Unknown~key~for~rowcolors }
9613    {
9614      Unknown~key.\\
9615      As~for~now,~there~is~only~two~keys~available~here:~'cols'~and~'respect-blocks'~
9616      (and~you~try~to~use~'\l_keys_key_str')\\
9617      That~key~will~be~ignored.
9618    }
9619  \@@_msg_new:nn { label~without~caption }
9620    {
9621      You~can't~use~the~key~'label'~in~your~'{NiceTabular}'~because~
9622      you~have~not~used~the~key~'caption'.~The~key~'label'~will~be~ignored.
9623    }
9624  \@@_msg_new:nn { W~warning }
9625    {
9626      Line~\msg_line_number:.~The~cell~is~too~wide~for~your~column~'W'~
9627      (row~\int_use:N \c@iRow).
9628    }
9629  \@@_msg_new:nn { Construct~too~large }
9630    {
9631      Construct~too~large.\\
9632      Your~command~\token_to_str:N #1
9633      can't~be~drawn~because~your~matrix~is~too~small.\\
9634      That~command~will~be~ignored.
9635    }
9636  \@@_msg_new:nn { underscore~after~nicematrix }
9637    {
9638      Problem~with~'underscore'.\\
9639      The~package~'underscore'~should~be~loaded~before~'nicematrix'.~
```

```
9640        You~can~go~on~but~you~won't~be~able~to~write~something~such~as:\\
9641        '\token_to_str:N \Cdots\token_to_str:N _{n-\token_to_str:N \text{~times}}'.
9642      }
9643  \@@_msg_new:nn { ampersand~in~light-syntax }
9644      {
9645        Ampersand~forbidden.\\
9646        You~can't~use~an~ampersand~(\token_to_str:N &)~to~separate~columns~because~
9647        ~the~key~'light-syntax'~is~in~force.~This~error~is~fatal.
9648      }
9649  \@@_msg_new:nn { double-backslash~in~light-syntax }
9650      {
9651        Double~backslash~forbidden.\\
9652        You~can't~use~\token_to_str:N
9653        \\~to~separate~rows~because~the~key~'light-syntax'~
9654        is~in~force.~You~must~use~the~character~'\l_@@_end_of_row_tl'~
9655        (set~by~the~key~'end-of-row').~This~error~is~fatal.
9656      }
9657  \@@_msg_new:nn { hlines~with~color }
9658      {
9659        Incompatible~keys.\\
9660        You~can't~use~the~keys~'hlines',~'vlines'~or~'hvlines'~for~a~
9661        '\token_to_str:N \Block'~when~the~key~'color'~or~'draw'~is~used.\\
9662        However,~you~can~put~several~commands~\token_to_str:N \Block.\\
9663        Your~key~will~be~discarded.
9664      }
9665  \@@_msg_new:nn { bad~value~for~baseline }
9666      {
9667        Bad~value~for~baseline.\\
9668        The~value~given~to~'baseline'~(\int_use:N \l_tmpa_int)~is~not~
9669        valid.~The~value~must~be~between~\int_use:N \l_@@_first_row_int\ and~
9670        \int_use:N \g_@@_row_total_int\ or~equal~to~'t',~'c'~or~'b'~or~of~
9671        the~form~'line-i'.\\
9672        A~value~of~1~will~be~used.
9673      }
9674  \@@_msg_new:nn { detection~of~empty~cells }
9675      {
9676        Problem~with~'not-empty'\\
9677        For~technical~reasons,~you~must~activate~
9678        'create-cell-nodes'~in~\token_to_str:N \CodeBefore\
9679        in~order~to~use~the~key~'\l_keys_key_str'.\\
9680        That~key~will~be~ignored.
9681      }
9682  \@@_msg_new:nn { siunitx~not~loaded }
9683      {
9684        siunitx~not~loaded\\
9685        You~can't~use~the~columns~'S'~because~'siunitx'~is~not~loaded.\\
9686        That~error~is~fatal.
9687      }
9688  \@@_msg_new:nn { ragged2e~not~loaded }
9689      {
9690        You~have~to~load~'ragged2e'~in~order~to~use~the~key~'\l_keys_key_str'~in~
9691        your~column~'\l_@@_vpos_col_str'~(or~'X').~The~key~'\str_lowercase:o
9692        \l_keys_key_str'~will~be~used~instead.
9693      }
9694  \@@_msg_new:nn { Invalid~name }
9695      {
9696        Invalid~name.\\
9697        You~can't~give~the~name~'\l_keys_value_tl'~to~a~\token_to_str:N
9698        \SubMatrix\ of~your~\@@_full_name_env:.\\
9699        A~name~must~be~accepted~by~the~regular~expression~[A-Za-z][A-Za-z0-9]*.\\
```

```
9700        This~key~will~be~ignored.
9701      }
9702    \@@_msg_new:nn { Wrong~line~in~SubMatrix }
9703      {
9704        Wrong~line.\\
9705        You~try~to~draw~a~#1~line~of~number~'#2'~in~a~
9706        \token_to_str:N \SubMatrix\ of~your~\@@_full_name_env:\ but~that~
9707        number~is~not~valid.~It~will~be~ignored.
9708      }
9709    \@@_msg_new:nn { Impossible~delimiter }
9710      {
9711        Impossible~delimiter.\\
9712        It's~impossible~to~draw~the~#1~delimiter~of~your~
9713        \token_to_str:N \SubMatrix\ because~all~the~cells~are~empty~
9714        in~that~column.
9715        \bool_if:NT \l_@@_submatrix_slim_bool
9716          { ~Maybe~you~should~try~without~the~key~'slim'. } \\
9717        This~\token_to_str:N \SubMatrix\ will~be~ignored.
9718      }
9719    \@@_msg_new:nnn { width~without~X~columns }
9720      {
9721        You~have~used~the~key~'width'~but~you~have~put~no~'X'~column.~
9722        That~key~will~be~ignored.
9723      }
9724      {
9725        This~message~is~the~message~'width~without~X~columns'~
9726        of~the~module~'nicematrix'.~
9727        The~experimented~users~can~disable~that~message~with~
9728        \token_to_str:N \msg_redirect_name:nnn.\\
9729      }
9730
9731    \@@_msg_new:nn { key~multiplicity~with~dotted }
9732      {
9733        Incompatible~keys. \\
9734        You~have~used~the~key~'multiplicity'~with~the~key~'dotted'~
9735        in~a~'custom-line'.~They~are~incompatible. \\
9736        The~key~'multiplicity'~will~be~discarded.
9737      }
9738    \@@_msg_new:nn { empty~environment }
9739      {
9740        Empty~environment.\\
9741        Your~\@@_full_name_env:\ is~empty.~This~error~is~fatal.
9742      }
9743    \@@_msg_new:nn { No~letter~and~no~command }
9744      {
9745        Erroneous~use.\\
9746        Your~use~of~'custom-line'~is~no-op~since~you~don't~have~used~the~
9747        key~'letter'~(for~a~letter~for~vertical~rules)~nor~the~keys~'command'~or~
9748        ~'ccommand'~(to~draw~horizontal~rules).\\
9749        However,~you~can~go~on.
9750      }
9751    \@@_msg_new:nn { Forbidden~letter }
9752      {
9753        Forbidden~letter.\\
9754        You~can't~use~the~letter~'#1'~for~a~customized~line.\\
9755        It~will~be~ignored.
9756      }
9757    \@@_msg_new:nn { Several~letters }
9758      {
9759        Wrong~name.\\
```

```
9760    You~must~use~only~one~letter~as~value~for~the~key~'letter'~(and~you~
9761    have~used~'\l_@@_letter_str').\\
9762    It~will~be~ignored.
9763  }
9764  \@@_msg_new:nn { Delimiter~with~small }
9765  {
9766    Delimiter~forbidden.\\
9767    You~can't~put~a~delimiter~in~the~preamble~of~your~\@@_full_name_env:\
9768    because~the~key~'small'~is~in~force.\\
9769    This~error~is~fatal.
9770  }
9771  \@@_msg_new:nn { unknown~cell~for~line~in~CodeAfter }
9772  {
9773    Unknown~cell.\\
9774    Your~command~\token_to_str:N\line\{#1\}\{#2\}~in~
9775    the~\token_to_str:N \CodeAfter\ of~your~\@@_full_name_env:\
9776    can't~be~executed~because~a~cell~doesn't~exist.\\
9777    This~command~\token_to_str:N \line\ will~be~ignored.
9778  }
9779  \@@_msg_new:nnn { Duplicate~name~for~SubMatrix }
9780  {
9781    Duplicate~name.\\
9782    The~name~'#1'~is~already~used~for~a~\token_to_str:N \SubMatrix\
9783    in~this~\@@_full_name_env:.\\
9784    This~key~will~be~ignored.\\
9785    \bool_if:NF \g_@@_messages_for_Overleaf_bool
9786      { For~a~list~of~the~names~already~used,~type~H~<return>. }
9787  }
9788  {
9789    The~names~already~defined~in~this~\@@_full_name_env:\ are:~
9790    \seq_use:Nnnn \g_@@_submatrix_names_seq { ~and~ } { ,~ } { ~and~ }.
9791  }
9792  \@@_msg_new:nn { r~or~l~with~preamble }
9793  {
9794    Erroneous~use.\\
9795    You~can't~use~the~key~'\l_keys_key_str'~in~your~\@@_full_name_env:.~
9796    You~must~specify~the~alignment~of~your~columns~with~the~preamble~of~
9797    your~\@@_full_name_env:.\\
9798    This~key~will~be~ignored.
9799  }
9800  \@@_msg_new:nn { Hdotsfor~in~col~0 }
9801  {
9802    Erroneous~use.\\
9803    You~can't~use~\token_to_str:N \Hdotsfor\ in~an~exterior~column~of~
9804    the~array.~This~error~is~fatal.
9805  }
9806  \@@_msg_new:nn { bad~corner }
9807  {
9808    Bad~corner.\\
9809    #1~is~an~incorrect~specification~for~a~corner~(in~the~key~
9810    'corners').~The~available~values~are:~NW,~SW,~NE~and~SE.\\
9811    This~specification~of~corner~will~be~ignored.
9812  }
9813  \@@_msg_new:nn { bad~border }
9814  {
9815    Bad~border.\\
9816    \l_keys_key_str\space~is~an~incorrect~specification~for~a~border~
9817    (in~the~key~'borders'~of~the~command~\token_to_str:N \Block).~
9818    The~available~values~are:~left,~right,~top~and~bottom~(and~you~can~
9819    also~use~the~key~'tikz'
9820    \IfPackageLoadedTF { tikz }
```

```
9821          { }
9822        {~if~you~load~the~LaTeX~package~'tikz'}).\\
9823      This~specification~of~border~will~be~ignored.
9824    }
9825  \@@_msg_new:nn { TikzEveryCell~without~tikz }
9826    {
9827      TikZ~not~loaded.\\
9828      You~can't~use~\token_to_str:N \TikzEveryCell\
9829      because~you~have~not~loaded~tikz.~
9830      This~command~will~be~ignored.
9831    }
9832  \@@_msg_new:nn { tikz~key~without~tikz }
9833    {
9834      TikZ~not~loaded.\\
9835      You~can't~use~the~key~'tikz'~for~the~command~'\token_to_str:N
9836      \Block'~because~you~have~not~loaded~tikz.~
9837      This~key~will~be~ignored.
9838    }
9839  \@@_msg_new:nn { last-col~non~empty~for~NiceArray }
9840    {
9841      Erroneous~use.\\
9842      In~the~\@@_full_name_env:,~you~must~use~the~key~
9843      'last-col'~without~value.\\
9844      However,~you~can~go~on~for~this~time~
9845      (the~value~'\l_keys_value_tl'~will~be~ignored).
9846    }
9847  \@@_msg_new:nn { last-col~non~empty~for~NiceMatrixOptions }
9848    {
9849      Erroneous~use.\\
9850      In~\token_to_str:N \NiceMatrixOptions,~you~must~use~the~key~
9851      'last-col'~without~value.\\
9852      However,~you~can~go~on~for~this~time~
9853      (the~value~'\l_keys_value_tl'~will~be~ignored).
9854    }
9855  \@@_msg_new:nn { Block~too~large~1 }
9856    {
9857      Block~too~large.\\
9858      You~try~to~draw~a~block~in~the~cell~#1-#2~of~your~matrix~but~the~matrix~is~
9859      too~small~for~that~block. \\
9860      This~block~and~maybe~others~will~be~ignored.
9861    }
9862  \@@_msg_new:nn { Block~too~large~2 }
9863    {
9864      Block~too~large.\\
9865      The~preamble~of~your~\@@_full_name_env:\ announces~\int_use:N
9866      \g_@@_static_num_of_col_int\
9867      columns~but~you~use~only~\int_use:N \c@jCol\ and~that's~why~a~block~
9868      specified~in~the~cell~#1-#2~can't~be~drawn.~You~should~add~some~ampersands~
9869      (&)~at~the~end~of~the~first~row~of~your~\@@_full_name_env:.\\
9870      This~block~and~maybe~others~will~be~ignored.
9871    }
9872  \@@_msg_new:nn { unknown~column~type }
9873    {
9874      Bad~column~type.\\
9875      The~column~type~'#1'~in~your~\@@_full_name_env:\
9876      is~unknown. \\
9877      This~error~is~fatal.
9878    }
9879  \@@_msg_new:nn { unknown~column~type~S }
9880    {
```

```
9881        Bad~column~type.\\
9882        The~column~type~'S'~in~your~\@@_full_name_env:\ is~unknown. \\
9883        If~you~want~to~use~the~column~type~'S'~of~siunitx,~you~should~
9884        load~that~package. \\
9885        This~error~is~fatal.
9886      }
9887    \@@_msg_new:nn { tabularnote~forbidden }
9888      {
9889        Forbidden~command.\\
9890        You~can't~use~the~command~\token_to_str:N\tabularnote\
9891        ~here.~This~command~is~available~only~in~
9892        \{NiceTabular\},~\{NiceTabular*\}~and~\{NiceTabularX\}~or~in~
9893        the~argument~of~a~command~\token_to_str:N \caption\ included~
9894        in~an~environment~{table}. \\
9895        This~command~will~be~ignored.
9896      }
9897    \@@_msg_new:nn { borders~forbidden }
9898      {
9899        Forbidden~key.\\
9900        You~can't~use~the~key~'borders'~of~the~command~\token_to_str:N \Block\
9901        because~the~option~'rounded-corners'~
9902        is~in~force~with~a~non-zero~value.\\
9903        This~key~will~be~ignored.
9904      }
9905    \@@_msg_new:nn { bottomrule~without~booktabs }
9906      {
9907        booktabs~not~loaded.\\
9908        You~can't~use~the~key~'tabular/bottomrule'~because~you~haven't~
9909        loaded~'booktabs'.\\
9910        This~key~will~be~ignored.
9911      }
9912    \@@_msg_new:nn { enumitem~not~loaded }
9913      {
9914        enumitem~not~loaded.\\
9915        You~can't~use~the~command~\token_to_str:N\tabularnote\
9916        ~because~you~haven't~loaded~'enumitem'.\\
9917        All~the~commands~\token_to_str:N\tabularnote\ will~be~
9918        ignored~in~the~document.
9919      }
9920    \@@_msg_new:nn { tikz~without~tikz }
9921      {
9922        Tikz~not~loaded.\\
9923        You~can't~use~the~key~'tikz'~here~because~Tikz~is~not~
9924        loaded.~If~you~go~on,~that~key~will~be~ignored.
9925      }
9926    \@@_msg_new:nn { tikz~in~custom-line~without~tikz }
9927      {
9928        Tikz~not~loaded.\\
9929        You~have~used~the~key~'tikz'~in~the~definition~of~a~
9930        customized~line~(with~'custom-line')~but~tikz~is~not~loaded.~
9931        You~can~go~on~but~you~will~have~another~error~if~you~actually~
9932        use~that~custom~line.
9933      }
9934    \@@_msg_new:nn { tikz~in~borders~without~tikz }
9935      {
9936        Tikz~not~loaded.\\
9937        You~have~used~the~key~'tikz'~in~a~key~'borders'~(of~a~
9938        command~'\token_to_str:N\Block')~but~tikz~is~not~loaded.~
9939        That~key~will~be~ignored.
9940      }
```

```
9941  \@@_msg_new:nn { without~color-inside }
9942    {
9943      If~order~to~use~\token_to_str:N \cellcolor,~\token_to_str:N \rowcolor,~
9944      \token_to_str:N \rowcolors\ or~\token_to_str:N \rowlistcolors\
9945      outside~\token_to_str:N \CodeBefore,~you~
9946      should~have~used~the~key~'color-inside'~in~your~\@@_full_name_env:.\\
9947      You~can~go~on~but~you~may~need~more~compilations.
9948    }
9949  \@@_msg_new:nn { color~in~custom-line~with~tikz }
9950    {
9951      Erroneous~use.\\
9952      In~a~'custom-line',~you~have~used~both~'tikz'~and~'color',~
9953      which~is~forbidden~(you~should~use~'color'~inside~the~key~'tikz').~
9954      The~key~'color'~will~be~discarded.
9955    }
9956  \@@_msg_new:nn { Wrong~last~row }
9957    {
9958      Wrong~number.\\
9959      You~have~used~'last-row=\int_use:N \l_@@_last_row_int'~but~your~
9960      \@@_full_name_env:\ seems~to~have~\int_use:N \c@iRow \ rows.~
9961      If~you~go~on,~the~value~of~\int_use:N \c@iRow \ will~be~used~for~
9962      last~row.~You~can~avoid~this~problem~by~using~'last-row'~
9963      without~value~(more~compilations~might~be~necessary).
9964    }
9965  \@@_msg_new:nn { Yet~in~env }
9966    {
9967      Nested~environments.\\
9968      Environments~of~nicematrix~can't~be~nested.\\
9969      This~error~is~fatal.
9970    }
9971  \@@_msg_new:nn { Outside~math~mode }
9972    {
9973      Outside~math~mode.\\
9974      The~\@@_full_name_env:\ can~be~used~only~in~math~mode~
9975      (and~not~in~\token_to_str:N \vcenter).\\
9976      This~error~is~fatal.
9977    }
9978  \@@_msg_new:nn { One~letter~allowed }
9979    {
9980      Bad~name.\\
9981      The~value~of~key~'\l_keys_key_str'~must~be~of~length~1.\\
9982      It~will~be~ignored.
9983    }
9984  \@@_msg_new:nn { TabularNote~in~CodeAfter }
9985    {
9986      Environment~{TabularNote}~forbidden.\\
9987      You~must~use~{TabularNote}~at~the~end~of~your~{NiceTabular}~
9988      but~*before*~the~\token_to_str:N \CodeAfter.\\
9989      This~environment~{TabularNote}~will~be~ignored.
9990    }
9991  \@@_msg_new:nn { varwidth~not~loaded }
9992    {
9993      varwidth~not~loaded.\\
9994      You~can't~use~the~column~type~'V'~because~'varwidth'~is~not~
9995      loaded.\\
9996      Your~column~will~behave~like~'p'.
9997    }
9998  \@@_msg_new:nnn { Unknow~key~for~RulesBis }
9999    {
10000     Unkown~key.\\
```

```
10001      Your~key~'\l_keys_key_str'~is~unknown~for~a~rule.\\
10002      \c_@@_available_keys_str
10003    }
10004    {
10005      The~available~keys~are~(in~alphabetic~order):~
10006      color,~
10007      dotted,~
10008      multiplicity,~
10009      sep-color,~
10010      tikz,~and~total-width.
10011    }
10012
10013  \@@_msg_new:nnn { Unknown~key~for~Block }
10014    {
10015      Unknown~key.\\
10016      The~key~'\l_keys_key_str'~is~unknown~for~the~command~\token_to_str:N
10017      \Block.\\ It~will~be~ignored. \\
10018      \c_@@_available_keys_str
10019    }
10020    {
10021      The~available~keys~are~(in~alphabetic~order):~&-in-blocks,~ampersand-in-blocks,~
10022      b,~B,~borders,~c,~draw,~fill,~hlines,~hvlines,~l,~line-width,~name,~
10023      opacity,~rounded-corners,~r,~respect-arraystretch,~t,~T,~tikz,~transparent~
10024      and~vlines.
10025    }
10026  \@@_msg_new:nnn { Unknown~key~for~Brace }
10027    {
10028      Unknown~key.\\
10029      The~key~'\l_keys_key_str'~is~unknown~for~the~commands~\token_to_str:N
10030      \UnderBrace\ and~\token_to_str:N \OverBrace.\\
10031      It~will~be~ignored. \\
10032      \c_@@_available_keys_str
10033    }
10034    {
10035      The~available~keys~are~(in~alphabetic~order):~color,~left-shorten,~
10036      right-shorten,~shorten~(which~fixes~both~left-shorten~and~
10037      right-shorten)~and~yshift.
10038    }
10039  \@@_msg_new:nnn { Unknown~key~for~CodeAfter }
10040    {
10041      Unknown~key.\\
10042      The~key~'\l_keys_key_str'~is~unknown.\\
10043      It~will~be~ignored. \\
10044      \c_@@_available_keys_str
10045    }
10046    {
10047      The~available~keys~are~(in~alphabetic~order):~
10048      delimiters/color,~
10049      rules~(with~the~subkeys~'color'~and~'width'),~
10050      sub-matrix~(several~subkeys)~
10051      and~xdots~(several~subkeys).~
10052      The~latter~is~for~the~command~\token_to_str:N \line.
10053    }
10054  \@@_msg_new:nnn { Unknown~key~for~CodeBefore }
10055    {
10056      Unknown~key.\\
10057      The~key~'\l_keys_key_str'~is~unknown.\\
10058      It~will~be~ignored. \\
10059      \c_@@_available_keys_str
10060    }
10061    {
10062      The~available~keys~are~(in~alphabetic~order):~
```

```
10063        create-cell-nodes,~
10064        delimiters/color~and~
10065        sub-matrix~(several~subkeys).
10066      }
10067   \@@_msg_new:nnn { Unknown~key~for~SubMatrix }
10068      {
10069        Unknown~key.\\
10070        The~key~'\l_keys_key_str'~is~unknown.\\
10071        That~key~will~be~ignored. \\
10072        \c_@@_available_keys_str
10073      }
10074      {
10075        The~available~keys~are~(in~alphabetic~order):~
10076        'delimiters/color',~
10077        'extra-height',~
10078        'hlines',~
10079        'hvlines',~
10080        'left-xshift',~
10081        'name',~
10082        'right-xshift',~
10083        'rules'~(with~the~subkeys~'color'~and~'width'),~
10084        'slim',~
10085        'vlines'~and~'xshift'~(which~sets~both~'left-xshift'~
10086        and~'right-xshift').\\
10087      }
10088   \@@_msg_new:nnn { Unknown~key~for~notes }
10089      {
10090        Unknown~key.\\
10091        The~key~'\l_keys_key_str'~is~unknown.\\
10092        That~key~will~be~ignored. \\
10093        \c_@@_available_keys_str
10094      }
10095      {
10096        The~available~keys~are~(in~alphabetic~order):~
10097        bottomrule,~
10098        code-after,~
10099        code-before,~
10100        detect-duplicates,~
10101        enumitem-keys,~
10102        enumitem-keys-para,~
10103        para,~
10104        label-in-list,~
10105        label-in-tabular~and~
10106        style.
10107      }
10108   \@@_msg_new:nnn { Unknown~key~for~RowStyle }
10109      {
10110        Unknown~key.\\
10111        The~key~'\l_keys_key_str'~is~unknown~for~the~command~
10112        \token_to_str:N \RowStyle. \\
10113        That~key~will~be~ignored. \\
10114        \c_@@_available_keys_str
10115      }
10116      {
10117        The~available~keys~are~(in~alphabetic~order):~
10118        'bold',~
10119        'cell-space-top-limit',~
10120        'cell-space-bottom-limit',~
10121        'cell-space-limits',~
10122        'color',~
10123        'nb-rows'~and~
10124        'rowcolor'.
10125      }
```

```
10126  \@@_msg_new:nnn { Unknown~key~for~NiceMatrixOptions }
10127    {
10128      Unknown~key.\\
10129      The~key~'\l_keys_key_str'~is~unknown~for~the~command~
10130      \token_to_str:N \NiceMatrixOptions. \\
10131      That~key~will~be~ignored. \\
10132      \c_@@_available_keys_str
10133    }
10134    {
10135      The~available~keys~are~(in~alphabetic~order):~
10136      &-in-blocks,~
10137      allow-duplicate-names,~
10138      ampersand-in-blocks,~
10139      caption-above,~
10140      cell-space-bottom-limit,~
10141      cell-space-limits,~
10142      cell-space-top-limit,~
10143      code-for-first-col,~
10144      code-for-first-row,~
10145      code-for-last-col,~
10146      code-for-last-row,~
10147      corners,~
10148      custom-key,~
10149      create-extra-nodes,~
10150      create-medium-nodes,~
10151      create-large-nodes,~
10152      custom-line,~
10153      delimiters~(several~subkeys),~
10154      end-of-row,~
10155      first-col,~
10156      first-row,~
10157      hlines,~
10158      hvlines,~
10159      hvlines-except-borders,~
10160      last-col,~
10161      last-row,~
10162      left-margin,~
10163      light-syntax,~
10164      light-syntax-expanded,~
10165      matrix/columns-type,~
10166      no-cell-nodes,~
10167      notes~(several~subkeys),~
10168      nullify-dots,~
10169      pgf-node-code,~
10170      renew-dots,~
10171      renew-matrix,~
10172      respect-arraystretch,~
10173      rounded-corners,~
10174      right-margin,~
10175      rules~(with~the~subkeys~'color'~and~'width'),~
10176      small,~
10177      sub-matrix~(several~subkeys),~
10178      vlines,~
10179      xdots~(several~subkeys).
10180    }
```

For '{NiceArray}', the set of keys is the same as for {NiceMatrix} excepted that there is no l and r.

```
10181  \@@_msg_new:nnn { Unknown~key~for~NiceArray }
10182    {
10183      Unknown~key.\\
10184      The~key~'\l_keys_key_str'~is~unknown~for~the~environment~
10185      \{NiceArray\}. \\
10186      That~key~will~be~ignored. \\
```

```
10187        \c_@@_available_keys_str
10188      }
10189      {
10190        The~available~keys~are~(in~alphabetic~order):~
10191        &-in-blocks,~
10192        ampersand-in-blocks,~
10193        b,~
10194        baseline,~
10195        c,~
10196        cell-space-bottom-limit,~
10197        cell-space-limits,~
10198        cell-space-top-limit,~
10199        code-after,~
10200        code-for-first-col,~
10201        code-for-first-row,~
10202        code-for-last-col,~
10203        code-for-last-row,~
10204        color-inside,~
10205        columns-width,~
10206        corners,~
10207        create-extra-nodes,~
10208        create-medium-nodes,~
10209        create-large-nodes,~
10210        extra-left-margin,~
10211        extra-right-margin,~
10212        first-col,~
10213        first-row,~
10214        hlines,~
10215        hvlines,~
10216        hvlines-except-borders,~
10217        last-col,~
10218        last-row,~
10219        left-margin,~
10220        light-syntax,~
10221        light-syntax-expanded,~
10222        name,~
10223        no-cell-nodes,~
10224        nullify-dots,~
10225        pgf-node-code,~
10226        renew-dots,~
10227        respect-arraystretch,~
10228        right-margin,~
10229        rounded-corners,~
10230        rules~(with~the~subkeys~'color'~and~'width'),~
10231        small,~
10232        t,~
10233        vlines,~
10234        xdots/color,~
10235        xdots/shorten-start,~
10236        xdots/shorten-end,~
10237        xdots/shorten~and~
10238        xdots/line-style.
10239      }
```

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`).

```
10240  \@@_msg_new:nnn { Unknown~key~for~NiceMatrix }
10241      {
10242        Unknown~key.\\
10243        The~key~'\l_keys_key_str'~is~unknown~for~the~
10244        \@@_full_name_env:. \\
10245        That~key~will~be~ignored. \\
10246        \c_@@_available_keys_str
10247      }
```

```
10248    {
10249      The~available~keys~are~(in~alphabetic~order):~
10250      &-in-blocks,~
10251      ampersand-in-blocks,~
10252      b,~
10253      baseline,~
10254      c,~
10255      cell-space-bottom-limit,~
10256      cell-space-limits,~
10257      cell-space-top-limit,~
10258      code-after,~
10259      code-for-first-col,~
10260      code-for-first-row,~
10261      code-for-last-col,~
10262      code-for-last-row,~
10263      color-inside,~
10264      columns-type,~
10265      columns-width,~
10266      corners,~
10267      create-extra-nodes,~
10268      create-medium-nodes,~
10269      create-large-nodes,~
10270      extra-left-margin,~
10271      extra-right-margin,~
10272      first-col,~
10273      first-row,~
10274      hlines,~
10275      hvlines,~
10276      hvlines-except-borders,~
10277      l,~
10278      last-col,~
10279      last-row,~
10280      left-margin,~
10281      light-syntax,~
10282      light-syntax-expanded,~
10283      name,~
10284      no-cell-nodes,~
10285      nullify-dots,~
10286      pgf-node-code,~
10287      r,~
10288      renew-dots,~
10289      respect-arraystretch,~
10290      right-margin,~
10291      rounded-corners,~
10292      rules~(with~the~subkeys~'color'~and~'width'),~
10293      small,~
10294      t,~
10295      vlines,~
10296      xdots/color,~
10297      xdots/shorten-start,~
10298      xdots/shorten-end,~
10299      xdots/shorten~and~
10300      xdots/line-style.
10301    }
10302  \@@_msg_new:nnn { Unknown~key~for~NiceTabular }
10303    {
10304      Unknown~key.\\
10305      The~key~'\l_keys_key_str'~is~unknown~for~the~environment~
10306      \{NiceTabular\}. \\
10307      That~key~will~be~ignored. \\
10308      \c_@@_available_keys_str
10309    }
10310    {
```

226

```
10311      The~available~keys~are~(in~alphabetic~order):~
10312      &-in-blocks,~
10313      ampersand-in-blocks,~
10314      b,~
10315      baseline,~
10316      c,~
10317      caption,~
10318      cell-space-bottom-limit,~
10319      cell-space-limits,~
10320      cell-space-top-limit,~
10321      code-after,~
10322      code-for-first-col,~
10323      code-for-first-row,~
10324      code-for-last-col,~
10325      code-for-last-row,~
10326      color-inside,~
10327      columns-width,~
10328      corners,~
10329      custom-line,~
10330      create-extra-nodes,~
10331      create-medium-nodes,~
10332      create-large-nodes,~
10333      extra-left-margin,~
10334      extra-right-margin,~
10335      first-col,~
10336      first-row,~
10337      hlines,~
10338      hvlines,~
10339      hvlines-except-borders,~
10340      label,~
10341      last-col,~
10342      last-row,~
10343      left-margin,~
10344      light-syntax,~
10345      light-syntax-expanded,~
10346      name,~
10347      no-cell-nodes,~
10348      notes~(several~subkeys),~
10349      nullify-dots,~
10350      pgf-node-code,~
10351      renew-dots,~
10352      respect-arraystretch,~
10353      right-margin,~
10354      rounded-corners,~
10355      rules~(with~the~subkeys~'color'~and~'width'),~
10356      short-caption,~
10357      t,~
10358      tabularnote,~
10359      vlines,~
10360      xdots/color,~
10361      xdots/shorten-start,~
10362      xdots/shorten-end,~
10363      xdots/shorten~and~
10364      xdots/line-style.
10365    }
10366 \@@_msg_new:nnn { Duplicate~name }
10367    {
10368      Duplicate~name.\\
10369      The~name~'\l_keys_value_tl'~is~already~used~and~you~shouldn't~use~
10370      the~same~environment~name~twice.~You~can~go~on,~but,~
10371      maybe,~you~will~have~incorrect~results~especially~
10372      if~you~use~'columns-width=auto'.~If~you~don't~want~to~see~this~
10373      message~again,~use~the~key~'allow-duplicate-names'~in~
```

```
10374        '\token_to_str:N \NiceMatrixOptions'.\\
10375      \bool_if:NF \g_@@_messages_for_Overleaf_bool
10376        { For~a~list~of~the~names~already~used,~type~H~<return>. }
10377    }
10378    {
10379      The~names~already~defined~in~this~document~are:~
10380      \seq_use:Nnnn \g_@@_names_seq { ~and~ } { ,~ } { ~and~ }.
10381    }
10382  \@@_msg_new:nn { Option~auto~for~columns-width }
10383    {
10384      Erroneous~use.\\
10385      You~can't~give~the~value~'auto'~to~the~key~'columns-width'~here.~
10386      That~key~will~be~ignored.
10387    }
10388  \@@_msg_new:nn { NiceTabularX~without~X }
10389    {
10390      NiceTabularX~without~X.\\
10391      You~should~not~use~{NiceTabularX}~without~X~columns.\\
10392      However,~you~can~go~on.
10393    }
10394  \@@_msg_new:nn { Preamble~forgotten }
10395    {
10396      Preamble~forgotten.\\
10397      You~have~probably~forgotten~the~preamble~of~your~
10398      \@@_full_name_env:. \\
10399      This~error~is~fatal.
10400    }
```

Contents