The code of the package nicematrix*

F. Pantigny
fpantigny@wanadoo.fr

May 29, 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 { \cs_if_exist_p:N \ar@ialign }
12 \bool_const:Nn \c_@@_testphase_table_bool
13   { \IfPackageLoadedTF { latex-lab-testphase-table } \c_true_bool \c_false_bool }
```

---

*This document corresponds to the version 6.28 of nicematrix, at the date of 2024/05/29.

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

```
22  \cs_new_protected:Npn \@@_msg_new:nnn #1 #2 #3
23    {
24      \bool_if:NTF \g_@@_messages_for_Overleaf_bool
25        { \msg_new:nnn { nicematrix } { #1 } { #2 \\ #3 } }
26        { \msg_new:nnnn { nicematrix } { #1 } { #2 } { #3 } }
27    }
```

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

```
28  \cs_new_protected:Npn \@@_error_or_warning:n
29    { \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".

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

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

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

```
55  \cs_set:Npn \int_if_zero:NT #1 { \int_compare:nNnT #1 = \c_zero_int }
56  \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.

```
57  \@@_msg_new:nn { Internal~error }
58    {
59      Potential~problem~when~using~nicematrix.\\
60      The~package~nicematrix~have~detected~a~modification~of~the~
61      standard~environment~{array}~(of~the~package~array).~Maybe~you~will~encounter~
62      some~slight~problems~when~using~nicematrix.~If~you~don't~want~to~see~
63      this~message~again,~load~nicematrix~with:~\token_to_str:N
64      \usepackage[no-test-for-array]{nicematrix}.
65    }
```

```
66  \@@_msg_new:nn { mdwtab~loaded }
67    {
68      The~packages~'mdwtab'~and~'nicematrix'~are~incompatible.~
69      This~error~is~fatal.
70    }
```

```
71  \cs_new_protected:Npn \@@_security_test:n #1
72    {
73      \peek_meaning:NTF \ignorespaces
74        { \@@_security_test_i:w }
75        { \@@_error:n { Internal~error } }
76      #1
77    }
```

```
78  \cs_new_protected:Npn \@@_security_test_i:w \ignorespaces #1
79    {
80      \peek_meaning:NF \unskip { \@@_error:n { Internal~error } }
81      #1
82    }
```

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

```
83  \hook_gput_code:nnn { begindocument / after } { . }
84    {
85      \IfPackageLoadedTF { mdwtab }
86        { \@@_fatal:n { mdwtab~loaded } }
87        {
88          \bool_if:NF \g_@@_no_test_for_array_bool
89            {
90              \group_begin:
91                \hbox_set:Nn \l_tmpa_box
92                  {
93                    \begin { tabular } { c > { \@@_security_test:n } c c }
94                    text & & text
95                    \end { tabular }
96                  }
97              \group_end:
```

```
 98            }
 99          }
100      }
```

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`).

```
101  \cs_new_protected:Npn \@@_collect_options:n #1
102    {
103      \peek_meaning:NTF [
104        { \@@_collect_options:nw { #1 } }
105        { #1 { } }
106    }
```

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

```
107  \NewDocumentCommand \@@_collect_options:nw { m r[] }
108    { \@@_collect_options:nn { #1 } { #2 } }
109
110  \cs_new_protected:Npn \@@_collect_options:nn #1 #2
111    {
112      \peek_meaning:NTF [
113        { \@@_collect_options:nnw { #1 } { #2 } }
114        { #1 { #2 } }
115    }
116
117  \cs_new_protected:Npn \@@_collect_options:nnw #1#2[#3]
118    { \@@_collect_options:nn { #1 } { #2 , #3 } }
```

4 Technical definitions

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

```
119  \tl_const:Nn \c_@@_b_tl { b }
120  \tl_const:Nn \c_@@_c_tl { c }
121  \tl_const:Nn \c_@@_l_tl { l }
122  \tl_const:Nn \c_@@_r_tl { r }
123  \tl_const:Nn \c_@@_all_tl { all }
124  \tl_const:Nn \c_@@_dot_tl { . }
125  \tl_const:Nn \c_@@_default_tl { default }
126  \tl_const:Nn \c_@@_star_tl { * }
127  \str_const:Nn \c_@@_r_str { r }
128  \str_const:Nn \c_@@_c_str { c }
129  \str_const:Nn \c_@@_l_str { l }
130  \str_const:Nn \c_@@_R_str { R }
```

```
131  \str_const:Nn \c_@@_C_str { C }
132  \str_const:Nn \c_@@_L_str { L }
133  \str_const:Nn \c_@@_j_str { j }
134  \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).

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

```
136  \cs_generate_variant:Nn \seq_set_split:Nnn { N o n }
137  \cs_generate_variant:Nn \str_lowercase:n { o }
```

```
138  \hook_gput_code:nnn { begindocument } { . }
139    {
140      \IfPackageLoadedTF { tikz }
141        {
```

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.

```
142          \tl_const:Nn \c_@@_pgfortikzpicture_tl { \exp_not:N \tikzpicture }
143          \tl_const:Nn \c_@@_endpgfortikzpicture_tl { \exp_not:N \endtikzpicture }
144        }
145        {
146          \tl_const:Nn \c_@@_pgfortikzpicture_tl { \exp_not:N \pgfpicture }
147          \tl_const:Nn \c_@@_endpgfortikzpicture_tl { \exp_not:N \endpgfpicture }
148        }
149    }
```

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).

```
150  \IfClassLoadedTF { revtex4-1 }
151    { \bool_const:Nn \c_@@_revtex_bool \c_true_bool }
152    {
153      \IfClassLoadedTF { revtex4-2 }
154        { \bool_const:Nn \c_@@_revtex_bool \c_true_bool }
155        {
```

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

```
156          \cs_if_exist:NT \rvtx@ifformat@geq
157            { \bool_const:Nn \c_@@_revtex_bool \c_true_bool }
158            { \bool_const:Nn \c_@@_revtex_bool \c_false_bool }
159        }
160    }
```

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.

```
161  \cs_new_protected:Npn \@@_provide_pgfsyspdfmark:
162    {
163      \iow_now:Nn \@mainaux
164        {
165          \ExplSyntaxOn
166          \cs_if_free:NT \pgfsyspdfmark
167            { \cs_set_eq:NN \pgfsyspdfmark \@gobblethree }
```

```
168          \ExplSyntaxOff
169        }
170      \cs_gset_eq:NN \@@_provide_pgfsyspdfmark: \prg_do_nothing:
171    }
```

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.

```
172  \ProvideDocumentCommand \iddots { }
173    {
174      \mathinner
175        {
176          \tex_mkern:D 1 mu
177          \box_move_up:nn { 1 pt } { \hbox { . } }
178          \tex_mkern:D 2 mu
179          \box_move_up:nn { 4 pt } { \hbox { . } }
180          \tex_mkern:D 2 mu
181          \box_move_up:nn { 7 pt }
182            { \vbox:n { \kern 7 pt \hbox { . } } }
183          \tex_mkern:D 1 mu
184        }
185    }
```

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.

```
186  \hook_gput_code:nnn { begindocument } { . }
187    {
188      \IfPackageLoadedTF { booktabs }
189        { \iow_now:Nn \@mainaux \nicematrix@redefine@check@rerun }
190        { }
191    }
192  \cs_set_protected:Npn \nicematrix@redefine@check@rerun
193    {
194      \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).

```
195      \cs_set_protected:Npn \pgfutil@check@rerun ##1 ##2
196        {
197          \str_if_eq:eeF { nm- } { \tl_range:nnn { ##1 } 1 3 }
198            { \@@_old_pgfutil@check@rerun { ##1 } { ##2 } }
199        }
200    }
```

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

```
201  \hook_gput_code:nnn { begindocument } { . }
202    {
203      \IfPackageLoadedTF { colortbl }
204        { }
205        {
```

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.

```
206          \cs_set_protected:Npn \CT@arc@ { }
207          \cs_set_nopar:Npn \arrayrulecolor #1 # { \CT@arc { #1 } }
208          \cs_set_nopar:Npn \CT@arc #1 #2
209            {
210              \dim_compare:nNnT \baselineskip = \c_zero_dim \noalign
```

```
211        { \cs_gset_nopar:Npn \CT@arc@ { \color #1 { #2 } } } }
212      }
```

Idem for `\CT@drs@`.

```
213      \cs_set_nopar:Npn \doublerulesepcolor #1 # { \CT@drs { #1 } }
214      \cs_set_nopar:Npn \CT@drs #1 #2
215        {
216          \dim_compare:nNnT \baselineskip = \c_zero_dim \noalign
217            { \cs_gset:Npn \CT@drsc@ { \color #1 { #2 } } } }
218        }
219      \cs_set_nopar:Npn \hline
220        {
221          \noalign { \ifnum 0 = `} \fi
222          \cs_set_eq:NN \hskip \vskip
223          \cs_set_eq:NN \vrule \hrule
224          \cs_set_eq:NN \@width \@height
225          { \CT@arc@ \vline }
226          \futurelet \reserved@a
227          \@xhline
228        }
229      }
230    }
```

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.

```
231  \cs_set_nopar:Npn \@@_standard_cline #1 { \@@_standard_cline:w #1 \q_stop }
232  \cs_set_nopar:Npn \@@_standard_cline:w #1-#2 \q_stop
233    {
234      \int_if_zero:nT \l_@@_first_col_int { \omit & }
235      \int_compare:nNnT { #1 } > \c_one_int
236        { \multispan { \int_eval:n { #1 - 1 } } & }
237      \multispan { \int_eval:n { #2 - #1 + 1 } }
238      {
239        \CT@arc@
240        \leaders \hrule \@height \arrayrulewidth \hfill
```

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

```
241        \skip_horizontal:N \c_zero_dim
242      }
```

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`.

```
243      \everycr { }
244      \cr
245      \noalign { \skip_vertical:N -\arrayrulewidth }
246    }
```

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.

```
247  \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`.

```
248    { \@@_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*.

---

[1]See question 99041 on TeX StackExchange.

```
249  \cs_set:Npn \@@_cline_i:nn #1 #2 { \@@_cline_i:w #1|#2- \q_stop }
250  \cs_set:Npn \@@_cline_i:w #1|#2-#3 \q_stop
251    {
252      \tl_if_empty:nTF { #3 }
253        { \@@_cline_iii:w #1|#2-#2 \q_stop }
254        { \@@_cline_ii:w #1|#2-#3 \q_stop }
255    }
256  \cs_set:Npn \@@_cline_ii:w #1|#2-#3-\q_stop
257    { \@@_cline_iii:w #1|#2-#3 \q_stop }
258  \cs_set:Npn \@@_cline_iii:w #1|#2-#3 \q_stop
259    {
```

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).

```
260      \int_compare:nNnT { #1 } < { #2 }
261        { \multispan { \int_eval:n { #2 - #1 } } & }
262      \multispan { \int_eval:n { #3 - #2 + 1 } }
263        {
264          \CT@arc@
265          \leaders \hrule \@height \arrayrulewidth \hfill
266          \skip_horizontal:N \c_zero_dim
267        }
```

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

```
268      \peek_meaning_remove_ignore_spaces:NTF \cline
269        { & \@@_cline_i:en { \int_eval:n { #3 + 1 } } }
270        { \everycr { } \cr }
271    }
272  \cs_generate_variant:Nn \@@_cline_i:nn { e n }
```

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

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

```
274  \cs_new_protected:Npn \@@_set_CT@arc@:n #1
275    {
276      \tl_if_blank:nF { #1 }
277        {
278          \tl_if_head_eq_meaning:nNTF { #1 } [
279            { \cs_set_nopar:Npn \CT@arc@ { \color #1 } }
280            { \cs_set_nopar:Npn \CT@arc@ { \color { #1 } } }
281        }
282    }
283  \cs_generate_variant:Nn \@@_set_CT@arc@:n { o }
```

```
284  \cs_new_protected:Npn \@@_set_CT@drsc@:n #1
285    {
286      \tl_if_head_eq_meaning:nNTF { #1 } [
287        { \cs_set_nopar:Npn \CT@drsc@ { \color #1 } }
288        { \cs_set_nopar:Npn \CT@drsc@ { \color { #1 } } }
289    }
290  \cs_generate_variant:Nn \@@_set_CT@drsc@:n { o }
```

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

```
291  \cs_new:Npn \@@_exp_color_arg:Nn #1 #2
292    {
293      \tl_if_head_eq_meaning:nNTF { #2 } [
294        { #1 #2 }
295        { #1 { #2 } }
296    }
297  \cs_generate_variant:Nn \@@_exp_color_arg:Nn { N o }
```

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

```
298 \cs_new_protected:Npn \@@_color:n #1
299   { \tl_if_blank:nF { #1 } { \@@_exp_color_arg:Nn \color { #1 } } }
300 \cs_generate_variant:Nn \@@_color:n { o }
```

```
301 \cs_new_protected:Npn \@@_rescan_for_spanish:N #1
302   {
303     \tl_set_rescan:Nno
304       #1
305       {
306         \char_set_catcode_other:N >
307         \char_set_catcode_other:N <
308       }
309       #1
310   }
```

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.

```
311 \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).

```
312 \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.

```
313 \NewExpandableDocumentCommand \NiceMatrixLastEnv { }
314   { \int_use:N \g_@@_env_int }
```

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

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

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

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

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

```
318 \bool_new:N \g_@@_delims_bool
319 \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.

```
320 \bool_new:N \l_@@_preamble_bool
321 \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.

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

The following counter will count the environments {NiceMatrixBlock}.

```
323 \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.

```
324 \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).

```
325 \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.).

```
326 \dim_new:N \l_@@_col_width_dim
327 \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.

```
328 \int_new:N \g_@@_row_total_int
329 \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).

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

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

```
331 \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.

```
332 \tl_new:N \l_@@_hpos_cell_tl
333 \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`.

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

Idem for the mono-row blocks.

```
335 \dim_new:N \g_@@_blocks_ht_dim
336 \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}).

```
337 \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`.

```
338 \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.

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

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

```
340 \bool_new:N \l_@@_notes_detect_duplicates_bool
341 \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.

```
342 \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).

```
343 \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).

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

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

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

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

```
346 \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.

```
347 \bool_new:N \l_@@_X_bool
348 \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 }`).

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

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

```
350 \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.

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

```
352 \tl_new:N \g_@@_left_delim_tl
353 \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}).

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

11

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

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

For `\multicolumn`.

```
356 \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`.

```
357 \tl_new:N \l_@@_columns_type_tl
358 \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 :.

```
359 \tl_new:N \l_@@_xdots_down_tl
360 \tl_new:N \l_@@_xdots_up_tl
361 \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`).

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

```
363 \cs_new_protected:Npn \@@_test_if_math_mode:
364   {
365     \if_mode_math: \else:
366       \@@_fatal:n { Outside~math~mode }
367     \fi:
368   }
```

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.

```
369 \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".

```
370 \colorlet { nicematrix-last-col } { . }
371 \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*).

```
372 \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*.

```
373 \tl_new:N \g_@@_com_or_env_str
374 \tl_gset:Nn \g_@@_com_or_env_str { environment }
```

```
375 \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).

```
376 \cs_new:Npn \@@_full_name_env:
377   {
378     \str_if_eq:onTF \g_@@_com_or_env_str { command }
379       { command \space \c_backslash_str \g_@@_name_env_str }
380       { environment \space \{ \g_@@_name_env_str \} }
381   }
```

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

```
382 \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.

```
383 \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.

```
384 \tl_new:N \g_@@_pre_code_before_tl
385 \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.

```
386 \tl_new:N \g_@@_pre_code_after_tl
387 \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.

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

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

```
389 \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.

```
390 \int_new:N \l_@@_old_iRow_int
391 \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).

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

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

```
393 \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.

```
394 \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.

```
395 \bool_new:N \l_@@_X_columns_aux_bool
396 \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.

```
397 \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).

```
398 \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).

```
399 \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`).

```
400 \tl_new:N \l_@@_code_before_tl
401 \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).

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

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

```
403 \dim_new:N \l_@@_x_initial_dim
404 \dim_new:N \l_@@_y_initial_dim
405 \dim_new:N \l_@@_x_final_dim
406 \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.

```
407 \dim_new:N \l_@@_tmpc_dim
408 \dim_new:N \l_@@_tmpd_dim
```

```
409 \dim_new:N \g_@@_dp_row_zero_dim
410 \dim_new:N \g_@@_ht_row_zero_dim
411 \dim_new:N \g_@@_ht_row_one_dim
412 \dim_new:N \g_@@_dp_ante_last_row_dim
413 \dim_new:N \g_@@_ht_last_row_dim
414 \dim_new:N \g_@@_dp_last_row_dim
```

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

```
415 \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".

```
416 \dim_new:N \g_@@_width_last_col_dim
417 \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.

```
418 \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!).

```
419 \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*}.

```
420 \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.

```
421 \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.

```
422 \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`).

```
423 \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.

```
424 \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).

```
425 \seq_new:N \g_@@_multicolumn_cells_seq
426 \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`).

```
427 \int_new:N \l_@@_row_min_int
428 \int_new:N \l_@@_row_max_int
429 \int_new:N \l_@@_col_min_int
430 \int_new:N \l_@@_col_max_int
```

The following counters will be used when drawing the rules.

```
431 \int_new:N \l_@@_start_int
432 \int_set_eq:NN \l_@@_start_int \c_one_int
433 \int_new:N \l_@@_end_int
434 \int_new:N \l_@@_local_start_int
435 \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.

```
436 \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).

```
437 \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`.

```
438 \tl_new:N \l_@@_fill_tl
439 \tl_new:N \l_@@_opacity_tl
440 \tl_new:N \l_@@_draw_tl
441 \seq_new:N \l_@@_tikz_seq
442 \clist_new:N \l_@@_borders_clist
443 \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.

```
444 \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`.

```
445 \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.

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

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

```
447 \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`).

```
448 \str_new:N \l_@@_hpos_block_str
449 \str_set:Nn \l_@@_hpos_block_str { c }
450 \bool_new:N \l_@@_hpos_of_block_cap_bool
451 \bool_new:N \l_@@_p_block_bool
```

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

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

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

```
453 \str_new:N \l_@@_vpos_block_str
454 \str_set:Nn \l_@@_vpos_block_str { c }
```

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

```
455 \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).

```
456 \bool_new:N \l_@@_vlines_block_bool
457 \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.

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

```
459 \dim_new:N \l_@@_submatrix_extra_height_dim
460 \dim_new:N \l_@@_submatrix_left_xshift_dim
461 \dim_new:N \l_@@_submatrix_right_xshift_dim
462 \clist_new:N \l_@@_hlines_clist
463 \clist_new:N \l_@@_vlines_clist
464 \clist_new:N \l_@@_submatrix_hlines_clist
465 \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}`).

```
466 \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.

```
467 \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`).

```
468 \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.

```
469     \int_new:N \l_@@_first_row_int
470     \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.

```
471     \int_new:N \l_@@_first_col_int
472     \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).

  ```
  473    \int_new:N \l_@@_last_row_int
  474    \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]

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

  Idem for `\l_@@_last_col_without_value_bool`

  ```
  476    \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.

  ```
  477    \int_new:N \l_@@_last_col_int
  478    \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.

  ```
  479    \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`).

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

Some utilities

```
481 \cs_set_protected:Npn \@@_cut_on_hyphen:w #1-#2\q_stop
482   {
483     \cs_set_nopar:Npn \l_tmpa_tl { #1 }
484     \cs_set_nopar:Npn \l_tmpb_tl { #2 }
485   }
```

---

[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.

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.

```
486  \cs_new_protected:Npn \@@_expand_clist:N #1
487    {
488      \clist_if_in:NVF #1 \c_@@_all_tl
489        {
490          \clist_clear:N \l_tmpa_clist
491          \clist_map_inline:Nn #1
492            {
493              \tl_if_in:nnTF { ##1 } { - }
494                { \@@_cut_on_hyphen:w ##1 \q_stop }
495                {
496                  \cs_set_nopar:Npn \l_tmpa_tl { ##1 }
497                  \cs_set_nopar:Npn \l_tmpb_tl { ##1 }
498                }
499              \int_step_inline:nnn \l_tmpa_tl \l_tmpb_tl
500                { \clist_put_right:Nn \l_tmpa_clist { ####1 } }
501            }
502          \tl_set_eq:NN #1 \l_tmpa_clist
503        }
504    }
```

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.

```
505  \hook_gput_code:nnn { begindocument } { . }
506    {
507      \dim_const:Nn \c_@@_shift_Ldots_last_row_dim { 0.5 em }
508      \dim_const:Nn \c_@@_shift_exterior_Vdots_dim { 0.6 em }
509      \dim_const:Nn \c_@@_innersep_middle_dim { 0.17 em }
510    }
```

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.

```
511 \newcounter { tabularnote }
```

```
512 \seq_new:N \g_@@_notes_seq
513 \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.

```
514 \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.

```
515 \seq_new:N \l_@@_notes_labels_seq
516 \newcounter{nicematrix_draft}
517 \cs_new_protected:Npn \@@_notes_format:n #1
518   {
519     \setcounter { nicematrix_draft } { #1 }
520     \@@_notes_style:n { nicematrix_draft }
521   }
```

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

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

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

```
523 \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`.

```
524 \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`.

```
525 \cs_set:Npn \thetabularnote { { \@@_notes_style:n { tabularnote } } }
```

---

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

```
526 \hook_gput_code:nnn { begindocument } { . }
527   {
528     \IfPackageLoadedTF { enumitem }
529       {
```

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.

```
530         \newlist { tabularnotes } { enumerate } { 1 }
531         \setlist [ tabularnotes ]
532           {
533             topsep = 0pt ,
534             noitemsep ,
535             leftmargin = * ,
536             align = left ,
537             labelsep = 0pt ,
538             label =
539               \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotesi } } ,
540           }
541         \newlist { tabularnotes* } { enumerate* } { 1 }
542         \setlist [ tabularnotes* ]
543           {
544             afterlabel = \nobreak ,
545             itemjoin = \quad ,
546             label =
547               \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotes*i } }
548           }
```

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.

```
549         \NewDocumentCommand \tabularnote { o m }
550           {
551             \bool_lazy_or:nnT { \cs_if_exist_p:N \@captype } \l_@@_in_env_bool
552               {
553                 \bool_lazy_and:nnTF { ! \l_@@_tabular_bool } \l_@@_in_env_bool
554                   { \@@_error:n { tabularnote~forbidden } }
555                   {
556                     \bool_if:NTF \l_@@_in_caption_bool
557                       \@@_tabularnote_caption:nn
558                       \@@_tabularnote:nn
559                     { #1 } { #2 }
560                   }
561               }
562           }
563       }
564       {
565         \NewDocumentCommand \tabularnote { o m }
566           {
567             \@@_error_or_warning:n { enumitem~not~loaded }
568             \@@_gredirect_none:n { enumitem~not~loaded }
569           }
570       }
571   }
572 \cs_new_protected:Npn \@@_test_first_novalue:nnn #1 #2 #3
573   { \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`.

```
574  \cs_new_protected:Npn \@@_tabularnote:nn #1 #2
575    {
```

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).

```
576    \int_zero:N \l_tmpa_int
577    \bool_if:NT \l_@@_notes_detect_duplicates_bool
578      {
```

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

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

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

```
579        \int_zero:N \l_tmpb_int
580        \seq_map_indexed_inline:Nn \g_@@_notes_seq
581          {
582            \@@_test_first_novalue:nnn ##2 { \int_incr:N \l_tmpb_int }
583            \tl_if_eq:nnT { { #1 } { #2 } } { ##2 }
584              {
585                \tl_if_novalue:nTF { #1 }
586                  { \int_set_eq:NN \l_tmpa_int \l_tmpb_int }
587                  { \int_set:Nn \l_tmpa_int { ##1 }  }
588                \seq_map_break:
589              }
590          }
591        \int_if_zero:nF \l_tmpa_int
592          { \int_add:Nn \l_tmpa_int \g_@@_notes_caption_int }
593      }
594    \int_if_zero:nT \l_tmpa_int
595      {
596        \seq_gput_right:Nn \g_@@_notes_seq { { #1 } { #2 } }
597        \tl_if_novalue:nT { #1 } { \int_gincr:N \c@tabularnote }
598      }
599    \seq_put_right:Nx \l_@@_notes_labels_seq
600      {
601        \tl_if_novalue:nTF { #1 }
602          {
603            \@@_notes_format:n
604              {
605                \int_eval:n
606                  {
607                    \int_if_zero:nTF \l_tmpa_int
608                      \c@tabularnote
609                      \l_tmpa_int
610                  }
611              }
612          }
613          { #1 }
614      }
615    \peek_meaning:NF \tabularnote
616      {
```

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.

```
617          \hbox_set:Nn \l_tmpa_box
618            {
```

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

```
619              \@@_notes_label_in_tabular:n
620                {
621                  \seq_use:Nnnn
622                    \l_@@_notes_labels_seq { , } { , } { , }
623                }
624            }
```

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

```
625          \int_gdecr:N \c@tabularnote
626          \int_set_eq:NN \l_tmpa_int \c@tabularnote
627          \refstepcounter { tabularnote }
628          \int_compare:nNnT \l_tmpa_int = \c@tabularnote
629            { \int_gincr:N \c@tabularnote }
630          \seq_clear:N \l_@@_notes_labels_seq
631          \bool_lazy_or:nnTF
632            { \tl_if_eq_p:NN \l_@@_hpos_cell_tl \c_@@_c_tl }
633            { \tl_if_eq_p:NN \l_@@_hpos_cell_tl \c_@@_r_tl }
634            {
635              \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).

```
636              \skip_horizontal:n { \box_wd:N \l_tmpa_box }
637            }
638            { \box_use:N \l_tmpa_box }
639        }
640    }
```

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`.

```
641 \cs_new_protected:Npn \@@_tabularnote_caption:nn #1 #2
642   {
643     \bool_if:NTF \g_@@_caption_finished_bool
644        {
645          \int_compare:nNnT \c@tabularnote = \g_@@_notes_caption_int
646            { \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`!

```
647          \seq_if_in:NnF \g_@@_notes_in_caption_seq { { #1 } { #2 } }
648            { \@@_error:n { Identical~notes~in~caption } }
649        }
650        {
```

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

```
651          \seq_if_in:NnTF \g_@@_notes_in_caption_seq { { #1 } { #2 } }
652            {
```

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.

```
653          \bool_gset_true:N \g_@@_caption_finished_bool
654          \int_gset_eq:NN \g_@@_notes_caption_int \c@tabularnote
655          \int_gzero:N \c@tabularnote
656        }
657        { \seq_gput_right:Nn \g_@@_notes_in_caption_seq { { #1 } { #2 } } }
658      }
```

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!

```
659      \tl_if_novalue:nT { #1 } { \int_gincr:N \c@tabularnote }
660      \seq_put_right:Nx \l_@@_notes_labels_seq
661        {
662          \tl_if_novalue:nTF { #1 }
663            { \@@_notes_format:n { \int_use:N \c@tabularnote } }
664            { #1 }
665        }
666      \peek_meaning:NF \tabularnote
667        {
668          \@@_notes_label_in_tabular:n
669            { \seq_use:Nnnn \l_@@_notes_labels_seq { , } { , } { , } }
670          \seq_clear:N \l_@@_notes_labels_seq
671        }
672    }
673  \cs_new_protected:Npn \@@_count_novalue_first:nn #1 #2
674    { \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.

```
675  \cs_new_protected:Npn \@@_pgf_rect_node:nnnnn #1 #2 #3 #4 #5
676    {
677      \begin { pgfscope }
678      \pgfset
679        {
680          inner~sep = \c_zero_dim ,
681          minimum~size = \c_zero_dim
682        }
683      \pgftransformshift { \pgfpoint { 0.5 * ( #2 + #4 ) } { 0.5 * ( #3 + #5 ) } }
684      \pgfnode
685        { rectangle }
686        { center }
687        {
688          \vbox_to_ht:nn
689            { \dim_abs:n { #5 - #3 } }
690            {
691              \vfill
692              \hbox_to_wd:nn { \dim_abs:n { #4 - #2 } } { }
693            }
694        }
695        { #1 }
696        { }
697      \end { pgfscope }
698    }
```

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.

```
699 \cs_new_protected:Npn \@@_pgf_rect_node:nnn #1 #2 #3
700   {
701     \begin { pgfscope }
702     \pgfset
703       {
704         inner~sep = \c_zero_dim ,
705         minimum~size = \c_zero_dim
706       }
707     \pgftransformshift { \pgfpointscale { 0.5 } { \pgfpointadd { #2 } { #3 } } }
708     \pgfpointdiff { #3 } { #2 }
709     \pgfgetlastxy \l_tmpa_dim \l_tmpb_dim
710     \pgfnode
711       { rectangle }
712       { center }
713       {
714         \vbox_to_ht:nn
715           { \dim_abs:n \l_tmpb_dim }
716           { \vfill \hbox_to_wd:nn { \dim_abs:n \l_tmpa_dim } { } }
717       }
718       { #1 }
719       { }
720     \end { pgfscope }
721   }
```

8   The options

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

```
722 \tl_new:N \l_@@_caption_tl
723 \tl_new:N \l_@@_short_caption_tl
724 \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).

```
725 \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.

```
726 \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`.

```
727 \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).

```
728 \dim_new:N \l_@@_cell_space_top_limit_dim
729 \dim_new:N \l_@@_cell_space_bottom_limit_dim
```

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

```
730 \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.

```
731 \dim_new:N \l_@@_xdots_inter_dim
732 \hook_gput_code:nnn { begindocument } { . }
733   { \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).

```
734 \dim_new:N \l_@@_xdots_shorten_start_dim
735 \dim_new:N \l_@@_xdots_shorten_end_dim
736 \hook_gput_code:nnn { begindocument } { . }
737   {
738     \dim_set:Nn \l_@@_xdots_shorten_start_dim { 0.3 em }
739     \dim_set:Nn \l_@@_xdots_shorten_end_dim { 0.3 em }
740   }
```

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.

```
741 \dim_new:N \l_@@_xdots_radius_dim
742 \hook_gput_code:nnn { begindocument } { . }
743   { \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.

```
744 \tl_new:N \l_@@_xdots_line_style_tl
745 \tl_const:Nn \c_@@_standard_tl { standard }
746 \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`.

```
747 \bool_new:N \l_@@_light_syntax_bool
748 \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).

```
749 \tl_new:N \l_@@_baseline_tl
750 \tl_set:Nn \l_@@_baseline_tl { c }
```

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

```
751 \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).

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

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

```
753 \bool_new:N \l_@@_parallelize_diags_bool
754 \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.

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

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

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

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

```
760 \cs_new_protected:Npn \@@_reset_arraystretch:
761   { \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).

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

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

```
763 \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.

```
764 \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".

```
765 \bool_new:N \l_@@_medium_nodes_bool
766 \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).

```
767 \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`).

```
768 \dim_new:N \l_@@_left_margin_dim
769 \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`.

```
770 \dim_new:N \l_@@_extra_left_margin_dim
771 \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.

```
772 \tl_new:N \l_@@_end_of_row_tl
773 \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 ":".

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

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

```
775 \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.

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

```
777 \keys_define:nn { NiceMatrix / xdots }
778   {
779     shorten-start .code:n =
780       \hook_gput_code:nnn { begindocument } { . }
781         { \dim_set:Nn \l_@@_xdots_shorten_start_dim { #1 } } ,
782     shorten-end .code:n =
783       \hook_gput_code:nnn { begindocument } { . }
784         { \dim_set:Nn \l_@@_xdots_shorten_end_dim { #1 } } ,
785     shorten-start .value_required:n = true ,
786     shorten-end .value_required:n = true ,
787     shorten .code:n =
788       \hook_gput_code:nnn { begindocument } { . }
789         {
790           \dim_set:Nn \l_@@_xdots_shorten_start_dim { #1 }
791           \dim_set:Nn \l_@@_xdots_shorten_end_dim { #1 }
792         } ,
793     shorten .value_required:n = true ,
794     horizontal-labels .bool_set:N = \l_@@_xdots_h_labels_bool ,
795     horizontal-labels .default:n = true ,
796     line-style .code:n =
797       {
798         \bool_lazy_or:nnTF
799           { \cs_if_exist_p:N \tikzpicture }
800           { \str_if_eq_p:nn { #1 } { standard } }
801           { \tl_set:Nn \l_@@_xdots_line_style_tl { #1 } }
802           { \@@_error:n { bad~option~for~line-style } }
803       } ,
804     line-style .value_required:n = true ,
805     color .tl_set:N = \l_@@_xdots_color_tl ,
806     color .value_required:n = true ,
807     radius .code:n =
808       \hook_gput_code:nnn { begindocument } { . }
809         { \dim_set:Nn \l_@@_xdots_radius_dim { #1 } } ,
810     radius .value_required:n = true ,
811     inter .code:n =
812       \hook_gput_code:nnn { begindocument } { . }
813         { \dim_set:Nn \l_@@_xdots_inter_dim { #1 } } ,
814     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 `^{...}`.

```
815      down .code:n = \tl_put_right:Nn \l_@@_xdots_down_tl { #1 } ,
816      up .code:n = \tl_put_right:Nn \l_@@_xdots_up_tl { #1 } ,
817      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).

```
818      draw-first .code:n = \prg_do_nothing: ,
819      unknown .code:n = \@@_error:n { Unknown~key~for~xdots }
820    }
```

```
821  \keys_define:nn { NiceMatrix / rules }
822    {
823      color .tl_set:N = \l_@@_rules_color_tl ,
824      color .value_required:n = true ,
825      width .dim_set:N = \arrayrulewidth ,
826      width .value_required:n = true ,
827      unknown .code:n = \@@_error:n { Unknown~key~for~rules }
828    }
```

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

```
829  \keys_define:nn { NiceMatrix / Global }
830    {
831      ampersand-in-blocks .bool_set:N = \l_@@_amp_in_blocks_bool ,
832      ampersand-in-blocks .default:n = true ,
833      no-cell-nodes .code:n =
834        \cs_set_protected:Npn \@@_node_for_cell:
835          { \box_use_drop:N \l_@@_cell_box } ,
836      no-cell-nodes .value_forbidden:n = true ,
837      rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
838      rounded-corners .default:n = 4 pt ,
839      custom-line .code:n = \@@_custom_line:n { #1 } ,
840      rules .code:n = \keys_set:nn { NiceMatrix / rules } { #1 } ,
841      rules .value_required:n = true ,
842      standard-cline .bool_set:N = \l_@@_standard_cline_bool ,
843      standard-cline .default:n = true ,
844      cell-space-top-limit .dim_set:N = \l_@@_cell_space_top_limit_dim ,
845      cell-space-top-limit .value_required:n = true ,
846      cell-space-bottom-limit .dim_set:N = \l_@@_cell_space_bottom_limit_dim ,
847      cell-space-bottom-limit .value_required:n = true ,
848      cell-space-limits .meta:n =
849        {
850          cell-space-top-limit = #1 ,
851          cell-space-bottom-limit = #1 ,
852        } ,
853      cell-space-limits .value_required:n = true ,
854      xdots .code:n = \keys_set:nn { NiceMatrix / xdots } { #1 } ,
855      light-syntax .code:n =
856        \bool_set_true:N \l_@@_light_syntax_bool
857        \bool_set_false:N \l_@@_light_syntax_expanded_bool ,
858      light-syntax .value_forbidden:n = true ,
859      light-syntax-expanded .code:n =
860        \bool_set_true:N \l_@@_light_syntax_bool
861        \bool_set_true:N \l_@@_light_syntax_expanded_bool ,
862      light-syntax-expanded .value_forbidden:n = true ,
863      end-of-row .tl_set:N = \l_@@_end_of_row_tl ,
864      end-of-row .value_required:n = true ,
865      first-col .code:n = \int_zero:N \l_@@_first_col_int ,
```

```
866    first-row .code:n = \int_zero:N \l_@@_first_row_int ,
867    last-row .int_set:N = \l_@@_last_row_int ,
868    last-row .default:n = -1 ,
869    code-for-first-col .tl_set:N = \l_@@_code_for_first_col_tl ,
870    code-for-first-col .value_required:n = true ,
871    code-for-last-col .tl_set:N = \l_@@_code_for_last_col_tl ,
872    code-for-last-col .value_required:n = true ,
873    code-for-first-row .tl_set:N = \l_@@_code_for_first_row_tl ,
874    code-for-first-row .value_required:n = true ,
875    code-for-last-row .tl_set:N = \l_@@_code_for_last_row_tl ,
876    code-for-last-row .value_required:n = true ,
877    hlines .clist_set:N = \l_@@_hlines_clist ,
878    vlines .clist_set:N = \l_@@_vlines_clist ,
879    hlines .default:n = all ,
880    vlines .default:n = all ,
881    vlines-in-sub-matrix .code:n =
882      {
883        \tl_if_single_token:nTF { #1 }
884          {
885            \tl_if_in:NnTF \c_@@_forbidden_letters_tl { #1 }
886              { \@@_error:nn { Forbidden~letter } { #1 } }
```

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

```
887            { \cs_set_eq:cN { @@ _ #1 } \@@_make_preamble_vlism:n }
888          }
889          { \@@_error:n { One~letter~allowed } }
890      } ,
891    vlines-in-sub-matrix .value_required:n = true ,
892    hvlines .code:n =
893      {
894        \bool_set_true:N \l_@@_hvlines_bool
895        \tl_set_eq:NN \l_@@_vlines_clist \c_@@_all_tl
896        \tl_set_eq:NN \l_@@_hlines_clist \c_@@_all_tl
897      } ,
898    hvlines-except-borders .code:n =
899      {
900        \tl_set_eq:NN \l_@@_vlines_clist \c_@@_all_tl
901        \tl_set_eq:NN \l_@@_hlines_clist \c_@@_all_tl
902        \bool_set_true:N \l_@@_hvlines_bool
903        \bool_set_true:N \l_@@_except_borders_bool
904      } ,
905    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.

```
906    renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
907    renew-dots .value_forbidden:n = true ,
908    nullify-dots .bool_set:N = \l_@@_nullify_dots_bool ,
909    create-medium-nodes .bool_set:N = \l_@@_medium_nodes_bool ,
910    create-large-nodes .bool_set:N = \l_@@_large_nodes_bool ,
911    create-extra-nodes .meta:n =
912      { create-medium-nodes , create-large-nodes } ,
913    left-margin .dim_set:N = \l_@@_left_margin_dim ,
914    left-margin .default:n = \arraycolsep ,
915    right-margin .dim_set:N = \l_@@_right_margin_dim ,
916    right-margin .default:n = \arraycolsep ,
917    margin .meta:n = { left-margin = #1 , right-margin = #1 } ,
918    margin .default:n = \arraycolsep ,
919    extra-left-margin .dim_set:N = \l_@@_extra_left_margin_dim ,
920    extra-right-margin .dim_set:N = \l_@@_extra_right_margin_dim ,
921    extra-margin .meta:n =
922      { extra-left-margin = #1 , extra-right-margin = #1 } ,
923    extra-margin .value_required:n = true ,
```

```
924    respect-arraystretch .code:n =
925      \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
926    respect-arraystretch .value_forbidden:n = true ,
927    pgf-node-code .tl_set:N = \l_@@_pgf_node_code_tl ,
928    pgf-node-code .value_required:n = true
929  }
```

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

```
930  \keys_define:nn { NiceMatrix / Env }
931    {
932      corners .clist_set:N = \l_@@_corners_clist ,
933      corners .default:n = { NW , SW , NE , SE } ,
934      code-before .code:n =
935        {
936          \tl_if_empty:nF { #1 }
937            {
938              \tl_gput_left:Nn \g_@@_pre_code_before_tl { #1 }
939              \bool_set_true:N \l_@@_code_before_bool
940            }
941        } ,
942      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}.

```
943      c .code:n = \tl_set:Nn \l_@@_baseline_tl c ,
944      t .code:n = \tl_set:Nn \l_@@_baseline_tl t ,
945      b .code:n = \tl_set:Nn \l_@@_baseline_tl b ,
946      baseline .tl_set:N = \l_@@_baseline_tl ,
947      baseline .value_required:n = true ,
948      columns-width .code:n =
949        \tl_if_eq:nnTF { #1 } { auto }
950          { \bool_set_true:N \l_@@_auto_columns_width_bool }
951          { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,
952      columns-width .value_required:n = true ,
953      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.

```
954        \legacy_if:nF { measuring@ }
955          {
956            \str_set:Nx \l_tmpa_str { #1 }
957            \seq_if_in:NVTF \g_@@_names_seq \l_tmpa_str
958              { \@@_error:nn { Duplicate~name } { #1 } }
959              { \seq_gput_left:NV \g_@@_names_seq \l_tmpa_str }
960            \str_set_eq:NN \l_@@_name_str \l_tmpa_str
961          } ,
962      name .value_required:n = true ,
963      code-after .tl_gset:N = \g_nicematrix_code_after_tl ,
964      code-after .value_required:n = true ,
965      color-inside .code:n =
966        \bool_set_true:N \l_@@_color_inside_bool
967        \bool_set_true:N \l_@@_code_before_bool ,
968      color-inside .value_forbidden:n = true ,
969      colortbl-like .meta:n = color-inside
970    }
971  \keys_define:nn { NiceMatrix / notes }
972    {
973      para .bool_set:N = \l_@@_notes_para_bool ,
974      para .default:n = true ,
975      code-before .tl_set:N = \l_@@_notes_code_before_tl ,
976      code-before .value_required:n = true ,
```

```
977    code-after .tl_set:N = \l_@@_notes_code_after_tl ,
978    code-after .value_required:n = true ,
979    bottomrule .bool_set:N = \l_@@_notes_bottomrule_bool ,
980    bottomrule .default:n = true ,
981    style .cs_set:Np = \@@_notes_style:n #1 ,
982    style .value_required:n = true ,
983    label-in-tabular .cs_set:Np = \@@_notes_label_in_tabular:n #1 ,
984    label-in-tabular .value_required:n = true ,
985    label-in-list .cs_set:Np = \@@_notes_label_in_list:n #1 ,
986    label-in-list .value_required:n = true ,
987    enumitem-keys .code:n =
988      {
989        \hook_gput_code:nnn { begindocument } { . }
990          {
991            \IfPackageLoadedTF { enumitem }
992              { \setlist* [ tabularnotes ] { #1 } }
993              { }
994          }
995      } ,
996    enumitem-keys .value_required:n = true ,
997    enumitem-keys-para .code:n =
998      {
999        \hook_gput_code:nnn { begindocument } { . }
1000         {
1001           \IfPackageLoadedTF { enumitem }
1002             { \setlist* [ tabularnotes* ] { #1 } }
1003             { }
1004         }
1005     } ,
1006    enumitem-keys-para .value_required:n = true ,
1007    detect-duplicates .bool_set:N = \l_@@_notes_detect_duplicates_bool ,
1008    detect-duplicates .default:n = true ,
1009    unknown .code:n  = \@@_error:n { Unknown~key~for~notes }
1010  }
1011 \keys_define:nn { NiceMatrix / delimiters }
1012  {
1013    max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1014    max-width .default:n = true ,
1015    color .tl_set:N = \l_@@_delimiters_color_tl ,
1016    color .value_required:n = true ,
1017  }
```

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

```
1018 \keys_define:nn { NiceMatrix }
1019  {
1020    NiceMatrixOptions .inherit:n =
1021      { NiceMatrix / Global } ,
1022    NiceMatrixOptions / xdots .inherit:n = NiceMatrix / xdots ,
1023    NiceMatrixOptions / rules .inherit:n = NiceMatrix / rules ,
1024    NiceMatrixOptions / notes .inherit:n = NiceMatrix / notes ,
1025    NiceMatrixOptions / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
1026    SubMatrix / rules .inherit:n = NiceMatrix / rules ,
1027    CodeAfter / xdots .inherit:n = NiceMatrix / xdots ,
1028    CodeBefore / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
1029    CodeAfter / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
1030    NiceMatrix .inherit:n =
1031      {
1032        NiceMatrix / Global ,
1033        NiceMatrix / Env ,
1034      } ,
1035    NiceMatrix / xdots .inherit:n = NiceMatrix / xdots ,
```

```
1036        NiceMatrix / rules .inherit:n = NiceMatrix / rules ,
1037        NiceTabular .inherit:n =
1038          {
1039            NiceMatrix / Global ,
1040            NiceMatrix / Env
1041          } ,
1042        NiceTabular / xdots .inherit:n = NiceMatrix / xdots ,
1043        NiceTabular / rules .inherit:n = NiceMatrix / rules ,
1044        NiceTabular / notes .inherit:n = NiceMatrix / notes ,
1045        NiceArray .inherit:n =
1046          {
1047            NiceMatrix / Global ,
1048            NiceMatrix / Env ,
1049          } ,
1050        NiceArray / xdots .inherit:n = NiceMatrix / xdots ,
1051        NiceArray / rules .inherit:n = NiceMatrix / rules ,
1052        pNiceArray .inherit:n =
1053          {
1054            NiceMatrix / Global ,
1055            NiceMatrix / Env ,
1056          } ,
1057        pNiceArray / xdots .inherit:n = NiceMatrix / xdots ,
1058        pNiceArray / rules .inherit:n = NiceMatrix / rules ,
1059      }
```

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

```
1060  \keys_define:nn { NiceMatrix / NiceMatrixOptions }
1061    {
1062      delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1063      delimiters / color .value_required:n = true ,
1064      delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1065      delimiters / max-width .default:n = true ,
1066      delimiters .code:n = \keys_set:nn { NiceMatrix / delimiters } { #1 } ,
1067      delimiters .value_required:n = true ,
1068      width .dim_set:N = \l_@@_width_dim ,
1069      width .value_required:n = true ,
1070      last-col .code:n =
1071        \tl_if_empty:nF { #1 }
1072          { \@@_error:n { last-col~non~empty~for~NiceMatrixOptions } }
1073        \int_zero:N \l_@@_last_col_int ,
1074      small .bool_set:N = \l_@@_small_bool ,
1075      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.

```
1076        renew-matrix .code:n = \@@_renew_matrix: ,
1077        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}.

```
1078        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.

```
1079        columns-width .code:n =
1080          \tl_if_eq:nnTF { #1 } { auto }
1081            { \@@_error:n { Option~auto~for~columns-width } }
1082            { \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.

```
1083    allow-duplicate-names .code:n =
1084      \@@_msg_redirect_name:nn { Duplicate~name } { none } ,
1085    allow-duplicate-names .value_forbidden:n = true ,
1086    notes .code:n = \keys_set:nn { NiceMatrix / notes } { #1 } ,
1087    notes .value_required:n = true ,
1088    sub-matrix .code:n = \keys_set:nn { NiceMatrix / sub-matrix } { #1 } ,
1089    sub-matrix .value_required:n = true ,
1090    matrix / columns-type .tl_set:N = \l_@@_columns_type_tl ,
1091    matrix / columns-type .value_required:n = true ,
1092    caption-above .bool_set:N = \l_@@_caption_above_bool ,
1093    caption-above .default:n = true ,
1094    unknown .code:n  = \@@_error:n { Unknown~key~for~NiceMatrixOptions }
1095  }
```

`\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.

```
1096  \NewDocumentCommand \NiceMatrixOptions { m }
1097    { \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.

```
1098  \keys_define:nn { NiceMatrix / NiceMatrix }
1099    {
1100      last-col .code:n = \tl_if_empty:nTF { #1 }
1101                        {
1102                          \bool_set_true:N \l_@@_last_col_without_value_bool
1103                          \int_set:Nn \l_@@_last_col_int { -1 }
1104                        }
1105                        { \int_set:Nn \l_@@_last_col_int { #1 } } ,
1106      columns-type .tl_set:N = \l_@@_columns_type_tl ,
1107      columns-type .value_required:n = true ,
1108      l .meta:n = { columns-type = l } ,
1109      r .meta:n = { columns-type = r } ,
1110      delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1111      delimiters / color .value_required:n = true ,
1112      delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1113      delimiters / max-width .default:n = true ,
1114      delimiters .code:n = \keys_set:nn { NiceMatrix / delimiters } { #1 } ,
1115      delimiters .value_required:n = true ,
1116      small .bool_set:N = \l_@@_small_bool ,
1117      small .value_forbidden:n = true ,
1118      unknown .code:n = \@@_error:n { Unknown~key~for~NiceMatrix }
1119    }
```

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

```
1120  \keys_define:nn { NiceMatrix / NiceArray }
1121    {
```

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.

```
1122      small .bool_set:N = \l_@@_small_bool ,
1123      small .value_forbidden:n = true ,
1124      last-col .code:n = \tl_if_empty:nF { #1 }
1125                        { \@@_error:n { last-col~non~empty~for~NiceArray } }
1126                        \int_zero:N \l_@@_last_col_int ,
1127      r .code:n = \@@_error:n { r~or~l~with~preamble } ,
1128      l .code:n = \@@_error:n { r~or~l~with~preamble } ,
```

```
1129    unknown .code:n = \@@_error:n { Unknown~key~for~NiceArray }
1130  }
1131 \keys_define:nn { NiceMatrix / pNiceArray }
1132  {
1133    first-col .code:n = \int_zero:N \l_@@_first_col_int ,
1134    last-col .code:n = \tl_if_empty:nF {#1}
1135                        { \@@_error:n { last-col~non~empty~for~NiceArray } }
1136                      \int_zero:N \l_@@_last_col_int ,
1137    first-row .code:n = \int_zero:N \l_@@_first_row_int ,
1138    delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1139    delimiters / color .value_required:n = true ,
1140    delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
1141    delimiters / max-width .default:n = true ,
1142    delimiters .code:n = \keys_set:nn { NiceMatrix / delimiters } { #1 } ,
1143    delimiters .value_required:n = true ,
1144    small .bool_set:N = \l_@@_small_bool ,
1145    small .value_forbidden:n = true ,
1146    r .code:n = \@@_error:n { r~or~l~with~preamble } ,
1147    l .code:n = \@@_error:n { r~or~l~with~preamble } ,
1148    unknown .code:n = \@@_error:n { Unknown~key~for~NiceMatrix }
1149  }
```

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

```
1150 \keys_define:nn { NiceMatrix / NiceTabular }
1151  {
```

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.

```
1152    width .code:n = \dim_set:Nn \l_@@_width_dim { #1 }
1153                   \bool_set_true:N \l_@@_width_used_bool ,
1154    width .value_required:n = true ,
1155    notes .code:n = \keys_set:nn { NiceMatrix / notes } { #1 } ,
1156    tabularnote .tl_gset:N = \g_@@_tabularnote_tl ,
1157    tabularnote .value_required:n = true ,
1158    caption .tl_set:N = \l_@@_caption_tl ,
1159    caption .value_required:n = true ,
1160    short-caption .tl_set:N = \l_@@_short_caption_tl ,
1161    short-caption .value_required:n = true ,
1162    label .tl_set:N = \l_@@_label_tl ,
1163    label .value_required:n = true ,
1164    last-col .code:n = \tl_if_empty:nF {#1}
1165                        { \@@_error:n { last-col~non~empty~for~NiceArray } }
1166                      \int_zero:N \l_@@_last_col_int ,
1167    r .code:n = \@@_error:n { r~or~l~with~preamble } ,
1168    l .code:n = \@@_error:n { r~or~l~with~preamble } ,
1169    unknown .code:n = \@@_error:n { Unknown~key~for~NiceTabular }
1170  }
```

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
```

```
1171 \keys_define:nn { NiceMatrix / CodeAfter }
1172  {
1173    delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1174    delimiters / color .value_required:n = true ,
1175    rules .code:n = \keys_set:nn { NiceMatrix / rules } { #1 } ,
1176    rules .value_required:n = true ,
```

```
1177    xdots .code:n = \keys_set:nn { NiceMatrix / xdots } { #1 } ,
1178    sub-matrix .code:n = \keys_set:nn { NiceMatrix / sub-matrix } { #1 } ,
1179    sub-matrix .value_required:n = true ,
1180    unknown .code:n = \@@_error:n { Unknown~key~for~CodeAfter }
1181  }
```

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}).

```
1182 \cs_new_protected:Npn \@@_cell_begin:w
1183   {
```

`\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.

```
1184    \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).

```
1185    \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
```

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

```
1186    \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.

```
1187    \int_compare:nNnT \c@jCol = \c_one_int
1188      { \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:`.

```
1189    \hbox_set:Nw \l_@@_cell_box
```

The following command is nullified in the tabulars.

```
1190    \@@_tuning_not_tabular_begin:

1191    \@@_tuning_first_row:
1192    \@@_tuning_last_row:
1193    \g_@@_row_style_tl
1194  }
```

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

```
1195 \cs_new_protected:Npn \@@_tuning_first_row:
1196   {
1197    \int_if_zero:nT \c@iRow
1198      {
1199        \int_compare:nNnT \c@jCol > \c_zero_int
1200          {
1201            \l_@@_code_for_first_row_tl
1202            \xglobal \colorlet { nicematrix-first-row } { . }
1203          }
1204      }
1205  }
```

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

```
1206 \cs_new_protected:Npn \@@_tuning_last_row:
1207   {
1208     \int_compare:nNnT \c@iRow = \l_@@_last_row_int
1209       {
1210         \l_@@_code_for_last_row_tl
1211         \xglobal \colorlet { nicematrix-last-row } { . }
1212       }
1213   }
```

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

```
1214 \cs_set_eq:NN \@@_tuning_key_small: \prg_do_nothing:
```

The following commands are nullified in the tabulars.

```
1215 \cs_set_nopar:Npn \@@_tuning_not_tabular_begin:
1216   {
1217     \c_math_toggle_token
```

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

```
1218     \@@_tuning_key_small:
1219   }
1220 \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.

```
1221 \cs_new_protected:Npn \@@_begin_of_row:
1222   {
1223     \int_gincr:N \c@iRow
1224     \dim_gset_eq:NN \g_@@_dp_ante_last_row_dim \g_@@_dp_last_row_dim
1225     \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
1226     \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1227     \pgfpicture
1228     \pgfrememberpicturepositiononpagetrue
1229     \pgfcoordinate
1230       { \@@_env: - row - \int_use:N \c@iRow - base }
1231       { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
1232     \str_if_empty:NF \l_@@_name_str
1233       {
1234         \pgfnodealias
1235           { \l_@@_name_str - row - \int_use:N \c@iRow - base }
1236           { \@@_env: - row - \int_use:N \c@iRow - base }
1237       }
1238     \endpgfpicture
1239   }
```

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.

```
1240 \cs_new_protected:Npn \@@_update_for_first_and_last_row:
1241   {
1242     \int_if_zero:nTF \c@iRow
1243       {
1244         \dim_gset:Nn \g_@@_dp_row_zero_dim
1245           { \dim_max:nn \g_@@_dp_row_zero_dim { \box_dp:N \l_@@_cell_box } }
1246         \dim_gset:Nn \g_@@_ht_row_zero_dim
1247           { \dim_max:nn \g_@@_ht_row_zero_dim { \box_ht:N \l_@@_cell_box } }
1248       }
1249       {
```

```
1250        \int_compare:nNnT \c@iRow = \c_one_int
1251          {
1252            \dim_gset:Nn \g_@@_ht_row_one_dim
1253              { \dim_max:nn \g_@@_ht_row_one_dim { \box_ht:N \l_@@_cell_box } }
1254          }
1255        }
1256    }
1257 \cs_new_protected:Npn \@@_rotate_cell_box:
1258    {
1259      \box_rotate:Nn \l_@@_cell_box { 90 }
1260      \bool_if:NTF \g_@@_rotate_c_bool
1261        {
1262          \hbox_set:Nn \l_@@_cell_box
1263            {
1264              \c_math_toggle_token
1265              \vcenter { \box_use:N \l_@@_cell_box }
1266              \c_math_toggle_token
1267            }
1268        }
1269        {
1270          \int_compare:nNnT \c@iRow = \l_@@_last_row_int
1271            {
1272              \vbox_set_top:Nn \l_@@_cell_box
1273                {
1274                  \vbox_to_zero:n { }
1275                  \skip_vertical:n { - \box_ht:N \@arstrutbox + 0.8 ex }
1276                  \box_use:N \l_@@_cell_box
1277                }
1278            }
1279        }
1280      \bool_gset_false:N \g_@@_rotate_bool
1281      \bool_gset_false:N \g_@@_rotate_c_bool
1282    }
1283 \cs_new_protected:Npn \@@_adjust_size_box:
1284    {
1285      \dim_compare:nNnT \g_@@_blocks_wd_dim > \c_zero_dim
1286        {
1287          \box_set_wd:Nn \l_@@_cell_box
1288            { \dim_max:nn { \box_wd:N \l_@@_cell_box } \g_@@_blocks_wd_dim }
1289          \dim_gzero:N \g_@@_blocks_wd_dim
1290        }
1291      \dim_compare:nNnT \g_@@_blocks_dp_dim > \c_zero_dim
1292        {
1293          \box_set_dp:Nn \l_@@_cell_box
1294            { \dim_max:nn { \box_dp:N \l_@@_cell_box } \g_@@_blocks_dp_dim }
1295          \dim_gzero:N \g_@@_blocks_dp_dim
1296        }
1297      \dim_compare:nNnT \g_@@_blocks_ht_dim > \c_zero_dim
1298        {
1299          \box_set_ht:Nn \l_@@_cell_box
1300            { \dim_max:nn { \box_ht:N \l_@@_cell_box } \g_@@_blocks_ht_dim }
1301          \dim_gzero:N \g_@@_blocks_ht_dim
1302        }
1303    }
1304 \cs_new_protected:Npn \@@_cell_end:
1305    {
```

The following command is nullified in the tabulars.

```
1306        \@@_tuning_not_tabular_end:
1307        \hbox_set_end:
1308        \@@_cell_end_i:
1309    }
```

```
1310 \cs_new_protected:Npn \@@_cell_end_i:
1311   {
```

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.

```
1312     \g_@@_cell_after_hook_tl
1313     \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
1314     \@@_adjust_size_box:
1315     \box_set_ht:Nn \l_@@_cell_box
1316       { \box_ht:N \l_@@_cell_box + \l_@@_cell_space_top_limit_dim }
1317     \box_set_dp:Nn \l_@@_cell_box
1318       { \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").

```
1319     \@@_update_max_cell_width:
```

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

```
1320     \@@_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.

```
1321     \bool_if:NTF \g_@@_empty_cell_bool
1322       { \box_use_drop:N \l_@@_cell_box }
1323       {
1324         \bool_if:NTF \g_@@_not_empty_cell_bool
1325           \@@_node_for_cell:
1326           {
1327             \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > \c_zero_dim
1328               \@@_node_for_cell:
1329               { \box_use_drop:N \l_@@_cell_box }
1330           }
1331       }
1332     \int_gset:Nn \g_@@_col_total_int { \int_max:nn \g_@@_col_total_int \c@jCol }
1333     \bool_gset_false:N \g_@@_empty_cell_bool
1334     \bool_gset_false:N \g_@@_not_empty_cell_bool
1335   }
```

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

```
1336 \cs_new_protected:Npn \@@_update_max_cell_width:
1337   {
1338     \dim_gset:Nn \g_@@_max_cell_width_dim
1339       { \dim_max:nn \g_@@_max_cell_width_dim { \box_wd:N \l_@@_cell_box } }
1340   }
```

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`).

```
1341 \cs_new_protected:Npn \@@_cell_end_for_w_s:
1342   {
1343     \@@_math_toggle:
1344     \hbox_set_end:
1345     \bool_if:NF \g_@@_rotate_bool
1346       {
1347         \hbox_set:Nn \l_@@_cell_box
1348           {
1349             \makebox [ \l_@@_col_width_dim ] [ s ]
1350               { \hbox_unpack_drop:N \l_@@_cell_box }
1351           }
1352       }
1353     \@@_cell_end_i:
1354   }
```

```
1355 \pgfset
1356   {
1357     nicematrix / cell-node /.style =
1358       {
1359         inner~sep = \c_zero_dim ,
1360         minimum~width = \c_zero_dim
1361       }
1362   }
```

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

```
1363 \cs_new_protected:Npn \@@_node_for_cell:
1364   {
1365     \pgfpicture
1366     \pgfsetbaseline \c_zero_dim
1367     \pgfrememberpicturepositiononpagetrue
1368     \pgfset { nicematrix / cell-node }
1369     \pgfnode
1370       { rectangle }
1371       { base }
1372       {
```

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).

```
1373         \set@color
1374         \box_use_drop:N \l_@@_cell_box
1375       }
1376     { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1377     { \l_@@_pgf_node_code_tl }
1378     \str_if_empty:NF \l_@@_name_str
1379       {
1380         \pgfnodealias
1381           { \l_@@_name_str - \int_use:N \c@iRow - \int_use:N \c@jCol }
1382           { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1383       }
1384     \endpgfpicture
1385   }
```

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.

```
1386 \cs_new_protected:Npn \@@_patch_node_for_cell:n #1
1387   {
1388     \cs_new_protected:Npn \@@_patch_node_for_cell:
1389       {
```

```
1390          \hbox_set:Nn \l_@@_cell_box
1391            {
1392              \box_move_up:nn { \box_ht:N \l_@@_cell_box}
1393              \hbox_overlap_left:n
1394                {
1395                  \pgfsys@markposition
1396                    { \@@_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.

```
1397                    #1
1398                }
1399              \box_use:N \l_@@_cell_box
1400              \box_move_down:nn { \box_dp:N \l_@@_cell_box }
1401              \hbox_overlap_left:n
1402                {
1403                  \pgfsys@markposition
1404                    { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - SE }
1405                  #1
1406                }
1407            }
1408          }
1409      }
```

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

```
1410  \bool_lazy_or:nnTF \sys_if_engine_xetex_p: \sys_if_output_dvi_p:
1411    {
1412      \@@_patch_node_for_cell:n
1413        { \skip_horizontal:n { 0.5 \box_wd:N \l_@@_cell_box } }
1414    }
1415    { \@@_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}
```
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).

```
1416  \cs_new_protected:Npn \@@_instruction_of_type:nnn #1 #2 #3
1417    {
1418      \bool_if:nTF { #1 } \tl_gput_left:cx \tl_gput_right:cx
1419        { g_@@_ #2 _ lines _ tl }
1420        {
1421          \use:c { @@ _ draw _ #2 : nnn }
1422            { \int_use:N \c@iRow }
1423            { \int_use:N \c@jCol }
1424            { \exp_not:n { #3 } } }
1425        }
1426    }
```

```
1427 \cs_new_protected:Npn \@@_array:
1428   {
1429 %    \begin{macrocode}
1430     \dim_set:Nn \col@sep
1431       { \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
1432     \dim_compare:nNnTF \l_@@_tabular_width_dim = \c_zero_dim
1433       { \cs_set_nopar:Npn \@halignto { } }
1434       { \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`.

```
1435     \@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.

```
1436     [ \str_if_eq:onTF \l_@@_baseline_tl c c t ]
1437   }
```

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`.

```
1438 \bool_if:NTF \c_@@_tagging_array_bool
1439   { \cs_set_eq:NN \@@_old_ar@ialign: \ar@ialign }
1440   { \cs_set_eq:NN \@@_old_ialign: \ialign }
```

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

```
1441 \cs_new_protected:Npn \@@_create_row_node:
1442   {
1443     \int_compare:nNnT \c@iRow > \g_@@_last_row_node_int
1444       {
1445         \int_gset_eq:NN \g_@@_last_row_node_int \c@iRow
1446         \@@_create_row_node_i:
1447       }
1448   }
1449 \cs_new_protected:Npn \@@_create_row_node_i:
1450   {
```

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

```
1451     \hbox
1452       {
1453         \bool_if:NT \l_@@_code_before_bool
1454           {
1455             \vtop
1456               {
1457                 \skip_vertical:N 0.5\arrayrulewidth
1458                 \pgfsys@markposition
1459                   { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1460                 \skip_vertical:N -0.5\arrayrulewidth
1461               }
1462           }
1463         \pgfpicture
1464         \pgfrememberpicturepositiononpagetrue
1465         \pgfcoordinate { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1466           { \pgfpoint \c_zero_dim { - 0.5 \arrayrulewidth } }
1467         \str_if_empty:NF \l_@@_name_str
1468           {
1469             \pgfnodealias
1470               { \l_@@_name_str - row - \int_eval:n { \c@iRow + 1 } }
1471               { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
1472           }
1473         \endpgfpicture
1474       }
1475   }
```

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

```
1476 \cs_new:Npn \@@_everycr: { \noalign { \@@_everycr_i: } }
```

```
1477 \cs_new_protected:Npn \@@_everycr_i:
1478   {
1479     \bool_if:NT \c_@@_testphase_table_bool
1480       {
1481         \tbl_if_row_was_started:T { \UseTaggingSocket { tbl / row / end } }
1482         \tbl_update_cell_data_for_next_row:
1483       }
1484     \int_gzero:N \c@jCol
1485     \bool_gset_false:N \g_@@_after_col_zero_bool
1486     \bool_if:NF \g_@@_row_of_col_done_bool
1487       {
1488         \@@_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).

```
1489         \tl_if_empty:NF \l_@@_hlines_clist
1490           {
1491             \tl_if_eq:NNF \l_@@_hlines_clist \c_@@_all_tl
1492               {
1493                 \exp_args:NNe
1494                   \clist_if_in:NnT
1495                   \l_@@_hlines_clist
1496                   { \int_eval:n { \c@iRow + 1 } }
1497               }
1498               {
```

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.

```
1499                 \int_compare:nNnT \c@iRow > { -1 }
1500                   {
1501                     \int_compare:nNnF \c@iRow = \l_@@_last_row_int
1502                       { \hrule height \arrayrulewidth width \c_zero_dim }
1503                   }
1504               }
1505           }
1506       }
1507   }
```

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

```
1508 \cs_set_protected:Npn \@@_renew_dots:
1509   {
1510     \cs_set_eq:NN \ldots \@@_Ldots
1511     \cs_set_eq:NN \cdots \@@_Cdots
1512     \cs_set_eq:NN \vdots \@@_Vdots
1513     \cs_set_eq:NN \ddots \@@_Ddots
1514     \cs_set_eq:NN \iddots \@@_Iddots
1515     \cs_set_eq:NN \dots \@@_Ldots
1516     \cs_set_eq:NN \hdotsfor \@@_Hdotsfor:
1517   }
```

```
1518 \cs_new_protected:Npn \@@_test_color_inside:
1519   {
1520     \bool_if:NF \l_@@_color_inside_bool
1521       {
```

We will issue an error only during the first run.

```
1522         \bool_if:NF \g_@@_aux_found_bool
1523           { \@@_error:n { without~color-inside } }
1524       }
1525   }
```

```
1526  \cs_new_protected:Npn \@@_redefine_everycr: { \everycr { \@@_everycr: } }
1527    \hook_gput_code:nnn { begindocument } { . }
1528      {
1529        \IfPackageLoadedTF { colortbl }
1530          {
1531            \cs_set_protected:Npn \@@_redefine_everycr:
1532              {
1533                \CT@everycr
1534                  {
1535                    \noalign { \cs_gset_eq:NN \CT@row@color \prg_do_nothing: }
1536                    \@@_everycr:
1537                  }
1538              }
1539          }
1540          { }
1541      }
```

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].

```
1542  \hook_gput_code:nnn { begindocument } { . }
1543    {
1544      \IfPackageLoadedTF { booktabs }
1545        {
1546          \cs_new_protected:Npn \@@_patch_booktabs:
1547            { \tl_put_left:Nn \@BTnormal \@@_create_row_node_i: }
1548        }
1549        { \cs_new_protected:Npn \@@_patch_booktabs: { } }
1550    }
```

The box `\@arstrutbox` is a box constructed in the beginning of the environment `{array}`. The construction of that box takes into account the current value of `\arraystretch`[5] and `\extrarowheight` (of `array`). That box is inserted (via `\@arstrut`) in the beginning of each row of the array. That's why we use the dimensions of that box to initialize the variables which will be the dimensions of the potential first and last row of the environment. This initialization must be done after the creation of `\@arstrutbox` and that's why we do it in the `\ialign`.

```
1551  \cs_new_protected:Npn \@@_some_initialization:
1552    {
1553      \dim_gset:Nn \g_@@_dp_row_zero_dim { \box_dp:N \@arstrutbox }
1554      \dim_gset:Nn \g_@@_ht_row_zero_dim { \box_ht:N \@arstrutbox }
1555      \dim_gset_eq:NN \g_@@_ht_row_one_dim \g_@@_ht_row_zero_dim
1556      \dim_gzero:N \g_@@_dp_ante_last_row_dim
1557      \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1558      \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
1559    }
```

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

```
1560  \cs_new_protected:Npn \@@_pre_array_ii:
1561    {
```

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

```
1562      \int_gzero:N \g_@@_total_X_weight_int
```

---

[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}`.

```
1563        \@@_expand_clist:N \l_@@_hlines_clist
1564        \@@_expand_clist:N \l_@@_vlines_clist
1565        \@@_patch_booktabs:
1566        \box_clear_new:N \l_@@_cell_box
1567        \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}`).

```
1568        \bool_if:NT \l_@@_small_bool
1569          {
1570            \cs_set_nopar:Npn \arraystretch { 0.47 }
1571            \dim_set:Nn \arraycolsep { 1.45 pt }
```

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

```
1572            \cs_set_eq:NN \@@_tuning_key_small: \scriptstyle
1573          }


1574        \bool_if:NT \g_@@_recreate_cell_nodes_bool
1575          {
1576            \tl_put_right:Nn \@@_begin_of_row:
1577              {
1578                \pgfsys@markposition
1579                  { \@@_env: - row - \int_use:N \c@iRow - base }
1580              }
1581          }
```

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`.

```
1582        \bool_if:NTF \c_@@_tagging_array_bool
1583          {
1584            \cs_set_nopar:Npn \ar@ialign
1585              {
1586                \bool_if:NT \c_@@_testphase_table_bool \tbl_init_cell_data_for_table:
1587                \@@_redefine_everycr:
1588                \dim_zero:N \tabskip
1589                \@@_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`.

```
1590                \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign:
1591                \halign
1592              }
1593          }
```

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).

```
1594          {
1595            \cs_set_nopar:Npn \ialign
1596              {
1597                \@@_redefine_everycr:
1598                \dim_zero:N \tabskip
1599                \@@_some_initialization:
1600                \cs_set_eq:NN \ialign \@@_old_ialign:
1601                \halign
1602              }
1603          }
```

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).

```
1604    \cs_set_eq:NN \@@_old_ldots \ldots
1605    \cs_set_eq:NN \@@_old_cdots \cdots
1606    \cs_set_eq:NN \@@_old_vdots \vdots
1607    \cs_set_eq:NN \@@_old_ddots \ddots
1608    \cs_set_eq:NN \@@_old_iddots \iddots
1609    \bool_if:NTF \l_@@_standard_cline_bool
1610      { \cs_set_eq:NN \cline \@@_standard_cline }
1611      { \cs_set_eq:NN \cline \@@_cline }
1612    \cs_set_eq:NN \Ldots \@@_Ldots
1613    \cs_set_eq:NN \Cdots \@@_Cdots
1614    \cs_set_eq:NN \Vdots \@@_Vdots
1615    \cs_set_eq:NN \Ddots \@@_Ddots
1616    \cs_set_eq:NN \Iddots \@@_Iddots
1617    \cs_set_eq:NN \Hline \@@_Hline:
1618    \cs_set_eq:NN \Hspace \@@_Hspace:
1619    \cs_set_eq:NN \Hdotsfor \@@_Hdotsfor:
1620    \cs_set_eq:NN \Vdotsfor \@@_Vdotsfor:
1621    \cs_set_eq:NN \Block \@@_Block:
1622    \cs_set_eq:NN \rotate \@@_rotate:
1623    \cs_set_eq:NN \OnlyMainNiceMatrix \@@_OnlyMainNiceMatrix:n
1624    \cs_set_eq:NN \dotfill \@@_dotfill:
1625    \cs_set_eq:NN \CodeAfter \@@_CodeAfter:
1626    \cs_set_eq:NN \diagbox \@@_diagbox:nn
1627    \cs_set_eq:NN \NotEmpty \@@_NotEmpty:
1628    \cs_set_eq:NN \RowStyle \@@_RowStyle:n
1629    \seq_map_inline:Nn \l_@@_custom_line_commands_seq
1630      { \cs_set_eq:cc { ##1 } { nicematrix - ##1 } }
1631    \cs_set_eq:NN \cellcolor \@@_cellcolor_tabular
1632    \cs_set_eq:NN \rowcolor \@@_rowcolor_tabular
1633    \cs_set_eq:NN \rowcolors \@@_rowcolors_tabular
1634    \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors_tabular
1635    \int_compare:nNnT \l_@@_first_row_int > \c_zero_int
1636      { \cs_set_eq:NN \@@_tuning_first_row: \prg_do_nothing: }
1637    \int_compare:nNnT \l_@@_last_row_int < \c_zero_int
1638      { \cs_set_eq:NN \@@_tuning_last_row: \prg_do_nothing: }
1639    \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:.

```
1640    \cs_set_eq:NN \multicolumn \@@_multicolumn:nnn
1641    \hook_gput_code:nnn { env / tabular / begin } { nicematrix }
1642      { \cs_set_eq:NN \multicolumn \@@_old_multicolumn }
1643    \@@_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!).

```
1644    \tl_if_exist:NT \l_@@_note_in_caption_tl
1645      {
1646        \tl_if_empty:NF \l_@@_note_in_caption_tl
1647          {
1648            \int_gset_eq:NN \g_@@_notes_caption_int \l_@@_note_in_caption_tl
1649            \int_gset:Nn \c@tabularnote { \l_@@_note_in_caption_tl }
1650          }
1651      }
```

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).

```
1652        \seq_gclear:N \g_@@_multicolumn_cells_seq
1653        \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).

```
1654        \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`).

```
1655        \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.

```
1656        \int_gzero_new:N \g_@@_col_total_int
1657        \cs_set_eq:NN \@ifnextchar \new@ifnextchar
1658        \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.

```
1659        \tl_gclear_new:N \g_@@_Cdots_lines_tl
1660        \tl_gclear_new:N \g_@@_Ldots_lines_tl
1661        \tl_gclear_new:N \g_@@_Vdots_lines_tl
1662        \tl_gclear_new:N \g_@@_Ddots_lines_tl
1663        \tl_gclear_new:N \g_@@_Iddots_lines_tl
1664        \tl_gclear_new:N \g_@@_HVdotsfor_lines_tl

1665        \tl_gclear:N \g_nicematrix_code_before_tl
1666        \tl_gclear:N \g_@@_pre_code_before_tl
1667     }
```

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

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

```
1668  \cs_new_protected:Npn \@@_pre_array:
1669    {
1670      \cs_if_exist:NT \theiRow { \int_set_eq:NN \l_@@_old_iRow_int \c@iRow }
1671      \int_gzero_new:N \c@iRow
1672      \cs_if_exist:NT \thejCol { \int_set_eq:NN \l_@@_old_jCol_int \c@jCol }
1673      \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).

```
1674      \int_compare:nNnT \l_@@_last_row_int = { -1 }
1675        {
1676          \bool_set_true:N \l_@@_last_row_without_value_bool
1677          \bool_if:NT \g_@@_aux_found_bool
1678            { \int_set:Nn \l_@@_last_row_int { \seq_item:Nn \g_@@_size_seq 3 } }
1679        }
1680      \int_compare:nNnT \l_@@_last_col_int = { -1 }
1681        {
1682          \bool_if:NT \g_@@_aux_found_bool
1683            { \int_set:Nn \l_@@_last_col_int { \seq_item:Nn \g_@@_size_seq 6 } }
1684        }
```

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".

```
1685        \int_compare:nNnT \l_@@_last_row_int > { -2 }
1686          {
1687            \tl_put_right:Nn \@@_update_for_first_and_last_row:
1688              {
1689                \dim_gset:Nn \g_@@_ht_last_row_dim
1690                  { \dim_max:nn \g_@@_ht_last_row_dim { \box_ht:N \l_@@_cell_box } }
1691                \dim_gset:Nn \g_@@_dp_last_row_dim
1692                  { \dim_max:nn \g_@@_dp_last_row_dim { \box_dp:N \l_@@_cell_box } }
1693              }
1694          }

1695        \seq_gclear:N \g_@@_cols_vlism_seq
1696        \seq_gclear:N \g_@@_submatrix_seq
```

Now the `\CodeBefore`.

```
1697        \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.

```
1698        \seq_gclear:N \g_@@_pos_of_blocks_seq
```

Idem for other sequences written on the aux file.

```
1699        \seq_gclear_new:N \g_@@_multicolumn_cells_seq
1700        \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 avoid warnings or, maybe, errors). That's why the command `\@@_create_row_node:` will use the following counter to avoid such construction.

```
1701        \int_gset:Nn \g_@@_last_row_node_int { -2 }
```

The value -2 is important.

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

```
1702        \@@_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.

```
1703        \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]`).

```
1704        \dim_zero_new:N \l_@@_left_delim_dim
1705        \dim_zero_new:N \l_@@_right_delim_dim
1706        \bool_if:NTF \g_@@_delims_bool
1707          {
```

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

```
1708            \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_left_delim_tl $ }
1709            \dim_set:Nn \l_@@_left_delim_dim { \box_wd:N \l_tmpa_box }
1710            \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_right_delim_tl $ }
1711            \dim_set:Nn \l_@@_right_delim_dim { \box_wd:N \l_tmpa_box }
1712          }
1713          {
1714            \dim_gset:Nn \l_@@_left_delim_dim
1715              { 2 \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
1716            \dim_gset_eq:NN \l_@@_right_delim_dim \l_@@_left_delim_dim
1717          }
```

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).

```
1718     \hbox_set:Nw \l_@@_the_array_box
1719     \bool_if:NT \c_@@_testphase_table_bool
1720       { \UseTaggingSocket { tbl / hmode / begin } }

1721     \skip_horizontal:N \l_@@_left_margin_dim
1722     \skip_horizontal:N \l_@@_extra_left_margin_dim
1723     \c_math_toggle_token
1724     \bool_if:NTF \l_@@_light_syntax_bool
1725       { \use:c { @@-light-syntax } }
1726       { \use:c { @@-normal-syntax } }
1727   }
```

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

```
1728 \cs_new_protected_nopar:Npn \@@_CodeBefore_Body:w #1 \Body
1729   {
1730     \tl_set:Nn \l_tmpa_tl { #1 }
1731     \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
1732       { \@@_rescan_for_spanish:N \l_tmpa_tl }
1733     \tl_gput_left:NV \g_@@_pre_code_before_tl \l_tmpa_tl
1734     \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...

```
1735     \@@_pre_array:
1736   }
```

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).

```
1737 \cs_new_protected:Npn \@@_pre_code_before:
1738   {
```

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).

```
1739     \int_set:Nn \c@iRow { \seq_item:Nn \g_@@_size_seq 2 }
1740     \int_set:Nn \c@jCol { \seq_item:Nn \g_@@_size_seq 5 }
1741     \int_set_eq:NN \g_@@_row_total_int { \seq_item:Nn \g_@@_size_seq 3 }
1742     \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.

```
1743     \pgfsys@markposition { \@@_env: - position }
1744     \pgfsys@getposition { \@@_env: - position } \@@_picture_position:
1745     \pgfpicture
1746     \pgf@relevantforpicturesizefalse
```

First, the recreation of the `row` nodes.

```
1747      \int_step_inline:nnn \l_@@_first_row_int { \g_@@_row_total_int + 1 }
1748        {
1749          \pgfsys@getposition { \@@_env: - row - ##1 } \@@_node_position:
1750          \pgfcoordinate { \@@_env: - row - ##1 }
1751            { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1752        }
```

Now, the recreation of the `col` nodes.

```
1753      \int_step_inline:nnn \l_@@_first_col_int { \g_@@_col_total_int + 1 }
1754        {
1755          \pgfsys@getposition { \@@_env: - col - ##1 } \@@_node_position:
1756          \pgfcoordinate { \@@_env: - col - ##1 }
1757            { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1758        }
```

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

```
1759      \@@_create_diag_nodes:
```

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

```
1760      \bool_if:NT \g_@@_recreate_cell_nodes_bool \@@_recreate_cell_nodes:
1761      \endpgfpicture
```

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

```
1762      \@@_create_blocks_nodes:
1763      \IfPackageLoadedTF { tikz }
1764        {
1765          \tikzset
1766            {
1767              every~picture / .style =
1768                { overlay , name~prefix = \@@_env: - }
1769            }
1770        }
1771        { }
1772      \cs_set_eq:NN \cellcolor \@@_cellcolor
1773      \cs_set_eq:NN \rectanglecolor \@@_rectanglecolor
1774      \cs_set_eq:NN \roundedrectanglecolor \@@_roundedrectanglecolor
1775      \cs_set_eq:NN \rowcolor \@@_rowcolor
1776      \cs_set_eq:NN \rowcolors \@@_rowcolors
1777      \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors
1778      \cs_set_eq:NN \arraycolor \@@_arraycolor
1779      \cs_set_eq:NN \columncolor \@@_columncolor
1780      \cs_set_eq:NN \chessboardcolors \@@_chessboardcolors
1781      \cs_set_eq:NN \SubMatrix \@@_SubMatrix_in_code_before
1782      \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
1783      \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
1784    }


1785 \cs_new_protected:Npn \@@_exec_code_before:
1786   {
1787      \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.

```
1788      \@@_add_to_colors_seq:nn { { nocolor } } { }
1789      \bool_gset_false:N \g_@@_recreate_cell_nodes_bool
1790      \group_begin:
```

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

```
1791      \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).

```
1792          \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
1793            { \@@_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.

```
1794          \exp_last_unbraced:NV \@@_CodeBefore_keys:
1795            \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.

```
1796          \@@_actually_color:
1797          \l_@@_code_before_tl
1798          \q_stop
1799        \bool_if:NT \l_@@_tabular_bool \c_math_toggle_token
1800        \group_end:
1801        \bool_if:NT \g_@@_recreate_cell_nodes_bool
1802          { \tl_put_left:Nn \@@_node_for_cell: \@@_patch_node_for_cell: }
1803      }


1804  \keys_define:nn { NiceMatrix / CodeBefore }
1805    {
1806      create-cell-nodes .bool_gset:N = \g_@@_recreate_cell_nodes_bool ,
1807      create-cell-nodes .default:n = true ,
1808      sub-matrix .code:n = \keys_set:nn { NiceMatrix / sub-matrix } { #1 } ,
1809      sub-matrix .value_required:n = true ,
1810      delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1811      delimiters / color .value_required:n = true ,
1812      unknown .code:n = \@@_error:n { Unknown~key~for~CodeBefore }
1813    }
1814  \NewDocumentCommand \@@_CodeBefore_keys: { O { } }
1815    {
1816      \keys_set:nn { NiceMatrix / CodeBefore } { #1 }
1817      \@@_CodeBefore:w
1818    }
```

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.

```
1819  \cs_new_protected:Npn \@@_CodeBefore:w #1 \q_stop
1820    {
1821      \bool_if:NT \g_@@_aux_found_bool
1822        {
1823          \@@_pre_code_before:
1824          #1
1825        }
1826    }
```

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.

```
1827  \cs_new_protected:Npn \@@_recreate_cell_nodes:
1828    {
1829      \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
```

```
1830        {
1831          \pgfsys@getposition { \@@_env: - ##1 - base } \@@_node_position:
1832          \pgfcoordinate { \@@_env: - row - ##1 - base }
1833            { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1834          \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
1835            {
1836              \cs_if_exist:cT
1837                { pgf @ sys @ pdf @ mark @ pos @ \@@_env: - ##1 - ####1 - NW }
1838                {
1839                  \pgfsys@getposition
1840                    { \@@_env: - ##1 - ####1 - NW }
1841                    \@@_node_position:
1842                  \pgfsys@getposition
1843                    { \@@_env: - ##1 - ####1 - SE }
1844                    \@@_node_position_i:
1845                  \@@_pgf_rect_node:nnn
1846                    { \@@_env: - ##1 - ####1 }
1847                    { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1848                    { \pgfpointdiff \@@_picture_position: \@@_node_position_i: }
1849                }
1850            }
1851        }
1852      \int_step_inline:nn \c@iRow
1853        {
1854          \pgfnodealias
1855            { \@@_env: - ##1 - last }
1856            { \@@_env: - ##1 - \int_use:N \c@jCol }
1857        }
1858      \int_step_inline:nn \c@jCol
1859        {
1860          \pgfnodealias
1861            { \@@_env: - last - ##1 }
1862            { \@@_env: - \int_use:N \c@iRow - ##1 }
1863        }
1864      \@@_create_extra_nodes:
1865    }


1866  \cs_new_protected:Npn \@@_create_blocks_nodes:
1867    {
1868      \pgfpicture
1869      \pgf@relevantforpicturesizefalse
1870      \pgfrememberpicturepositiononpagetrue
1871      \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
1872        { \@@_create_one_block_node:nnnnn ##1 }
1873      \endpgfpicture
1874    }
```

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]

```
1875  \cs_new_protected:Npn \@@_create_one_block_node:nnnnn #1 #2 #3 #4 #5
1876    {
1877      \tl_if_empty:nF { #5 }
1878        {
1879          \@@_qpoint:n { col - #2 }
1880          \dim_set_eq:NN \l_tmpa_dim \pgf@x
1881          \@@_qpoint:n { #1 }
1882          \dim_set_eq:NN \l_tmpb_dim \pgf@y
1883          \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
1884          \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
```

---

[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).

```
1885        \@@_qpoint:n { \int_eval:n { #3 + 1 } }
1886        \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
1887        \@@_pgf_rect_node:nnnnn
1888          { \@@_env: - #5 }
1889          { \dim_use:N \l_tmpa_dim }
1890          { \dim_use:N \l_tmpb_dim }
1891          { \dim_use:N \l_@@_tmpc_dim }
1892          { \dim_use:N \l_@@_tmpd_dim }
1893      }
1894    }


1895  \cs_new_protected:Npn \@@_patch_for_revtex:
1896    {
1897      \cs_set_eq:NN \@addamp \@addamp@LaTeX
1898      \cs_set_eq:NN \insert@column \insert@column@array
1899      \cs_set_eq:NN \@classx \@classx@array
1900      \cs_set_eq:NN \@xarraycr \@xarraycr@array
1901      \cs_set_eq:NN \@arraycr \@arraycr@array
1902      \cs_set_eq:NN \@xargarraycr \@xargarraycr@array
1903      \cs_set_eq:NN \array \array@array
1904      \cs_set_eq:NN \@array \@array@array
1905      \cs_set_eq:NN \@tabular \@tabular@array
1906      \cs_set_eq:NN \@mkpream \@mkpream@array
1907      \cs_set_eq:NN \endarray \endarray@array
1908      \cs_set:Npn \@tabarray { \@ifnextchar [ { \@array } { \@array [ c ] } }
1909      \cs_set:Npn \endtabular { \endarray $\egroup} % $
1910    }
```

11   The environment {NiceArrayWithDelims}

```
1911  \NewDocumentEnvironment { NiceArrayWithDelims }
1912    { m m O { } m ! O { } t \CodeBefore }
1913    {
1914      \bool_if:NT \c_@@_revtex_bool \@@_patch_for_revtex:

1915      \@@_provide_pgfsyspdfmark:
1916      \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.

```
1917      \bgroup

1918      \tl_gset:Nn \g_@@_left_delim_tl { #1 }
1919      \tl_gset:Nn \g_@@_right_delim_tl { #2 }
1920      \tl_gset:Nn \g_@@_user_preamble_tl { #4 }
1921      \tl_if_empty:NT \g_@@_user_preamble_tl { \@@_fatal:n { empty~preamble } }

1922      \int_gzero:N \g_@@_block_box_int
1923      \dim_zero:N \g_@@_width_last_col_dim
1924      \dim_zero:N \g_@@_width_first_col_dim
1925      \bool_gset_false:N \g_@@_row_of_col_done_bool
1926      \str_if_empty:NT \g_@@_name_env_str
1927        { \str_gset:Nn \g_@@_name_env_str { NiceArrayWithDelims } }
1928      \bool_if:NTF \l_@@_tabular_bool
1929        \mode_leave_vertical:
1930        \@@_test_if_math_mode:
1931      \bool_if:NT \l_@@_in_env_bool { \@@_fatal:n { Yet~in~env } }
1932      \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.

```
1933        \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.

```
1934        \cs_if_exist:NT \tikz@library@external@loaded
1935          {
1936            \tikzexternaldisable
1937            \cs_if_exist:NT \ifstandalone
1938              { \tikzset { external / optimize = false } }
1939          }
```

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

```
1940        \int_gincr:N \g_@@_env_int
1941        \bool_if:NF \l_@@_block_auto_columns_width_bool
1942          { \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).

```
1943        \seq_gclear:N \g_@@_blocks_seq
1944        \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`.

```
1945        \seq_gclear:N \g_@@_pos_of_stroken_blocks_seq
1946        \seq_gclear:N \g_@@_pos_of_xdots_seq
1947        \tl_gclear_new:N \g_@@_code_before_tl
1948        \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.

```
1949        \tl_if_exist:cTF { c_@@ _ \int_use:N \g_@@_env_int _ tl }
1950          {
1951            \bool_gset_true:N \g_@@_aux_found_bool
1952            \use:c { c_@@ _ \int_use:N \g_@@_env_int _ tl }
1953          }
1954          { \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.

```
1955        \tl_gclear:N \g_@@_aux_tl
1956        \tl_if_empty:NF \g_@@_code_before_tl
1957          {
1958            \bool_set_true:N \l_@@_code_before_bool
1959            \tl_put_right:NV \l_@@_code_before_tl \g_@@_code_before_tl
1960          }
1961        \tl_if_empty:NF \g_@@_pre_code_before_tl
1962          { \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.

```
1963        \bool_if:NTF \g_@@_delims_bool
1964          { \keys_set:nn { NiceMatrix / pNiceArray } }
1965          { \keys_set:nn { NiceMatrix / NiceArray } }
1966        { #3 , #5 }
```

---

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

```
1967        \@@_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:.

```
1968        \bool_if:nTF { #6 } \@@_CodeBefore_Body:w \@@_pre_array:
1969      }
```

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

```
1970      {
1971        \bool_if:NTF \l_@@_light_syntax_bool
1972          { \use:c { end @@-light-syntax } }
1973          { \use:c { end @@-normal-syntax } }
1974        \c_math_toggle_token
1975        \skip_horizontal:N \l_@@_right_margin_dim
1976        \skip_horizontal:N \l_@@_extra_right_margin_dim
1977
1978        % % awful workaround
1979        \int_compare:nNnT \g_@@_col_total_int = \c_one_int
1980          {
1981            \dim_compare:nNnT \l_@@_columns_width_dim > \c_zero_dim
1982              {
1983                \skip_horizontal:N - \l_@@_columns_width_dim
1984                \bool_if:NTF \l_@@_tabular_bool
1985                  { \skip_horizontal:n { - 2 \tabcolsep } }
1986                  { \skip_horizontal:n { - 2 \arraycolsep } }
1987              }
1988          }
1989        \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.

```
1990        \bool_if:NT \l_@@_width_used_bool
1991          {
1992            \int_if_zero:nT \g_@@_total_X_weight_int
1993              { \@@_error_or_warning:n { width~without~X~columns } }
1994          }
```

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.

```
1995        \int_compare:nNnT \g_@@_total_X_weight_int > \c_zero_int
1996          {
1997            \tl_gput_right:Nx \g_@@_aux_tl
1998              {
1999                \bool_set_true:N \l_@@_X_columns_aux_bool
2000                \dim_set:Nn \l_@@_X_columns_dim
2001                  {
2002                    \dim_compare:nNnTF
2003                      {
2004                        \dim_abs:n
2005                          { \l_@@_width_dim - \box_wd:N \l_@@_the_array_box }
2006                      }
2007                    <
2008                    { 0.001 pt }
2009                    { \dim_use:N \l_@@_X_columns_dim }
2010                    {
2011                      \dim_eval:n
2012                        {
```

```
2013                    ( \l_@@_width_dim - \box_wd:N \l_@@_the_array_box )
2014                    / \int_use:N \g_@@_total_X_weight_int
2015                    + \l_@@_X_columns_dim
2016                  }
2017                }
2018              }
2019            }
2020          }
```

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).

```
2021        \int_compare:nNnT \l_@@_last_row_int > { -2 }
2022          {
2023            \bool_if:NF \l_@@_last_row_without_value_bool
2024              {
2025                \int_compare:nNnF \l_@@_last_row_int = \c@iRow
2026                  {
2027                    \@@_error:n { Wrong~last~row }
2028                    \int_gset_eq:NN \l_@@_last_row_int \c@iRow
2029                  }
2030              }
2031          }
```

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]

```
2032        \int_gset_eq:NN \c@jCol \g_@@_col_total_int
2033        \bool_if:NTF \g_@@_last_col_found_bool
2034          { \int_gdecr:N \c@jCol }
2035          {
2036            \int_compare:nNnT \l_@@_last_col_int > { -1 }
2037              { \@@_error:n { last~col~not~used } }
2038          }
```

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

```
2039        \int_gset_eq:NN \g_@@_row_total_int \c@iRow
2040        \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.).

```
2041        \int_if_zero:nT \l_@@_first_col_int
2042          { \skip_horizontal:N \g_@@_width_first_col_dim }
```

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

```
2043        \bool_if:nTF { ! \g_@@_delims_bool }
2044          {
2045            \tl_if_eq:NNTF \l_@@_baseline_tl \c_@@_c_tl
2046              \@@_use_arraybox_with_notes_c:
2047              {
2048                \tl_if_eq:NNTF \l_@@_baseline_tl \c_@@_b_tl
2049                  \@@_use_arraybox_with_notes_b:
2050                  \@@_use_arraybox_with_notes:
2051              }
2052          }
```

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).

```
2053          {
2054            \int_if_zero:nTF \l_@@_first_row_int
2055              {
2056                \dim_set_eq:NN \l_tmpa_dim \g_@@_dp_row_zero_dim
```

---

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

```
2057          \dim_add:Nn \l_tmpa_dim \g_@@_ht_row_zero_dim
2058        }
2059        { \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]

```
2060        \int_compare:nNnTF \l_@@_last_row_int > { -2 }
2061        {
2062          \dim_set_eq:NN \l_tmpb_dim \g_@@_ht_last_row_dim
2063          \dim_add:Nn \l_tmpb_dim \g_@@_dp_last_row_dim
2064        }
2065        { \dim_zero:N \l_tmpb_dim }
2066      \hbox_set:Nn \l_tmpa_box
2067        {
2068          \c_math_toggle_token
2069          \@@_color:o \l_@@_delimiters_color_tl
2070          \exp_after:wN \left \g_@@_left_delim_tl
2071          \vcenter
2072            {
```

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.

```
2073              \skip_vertical:n { -\l_tmpa_dim - \arrayrulewidth }
2074              \hbox
2075                {
2076                  \bool_if:NTF \l_@@_tabular_bool
2077                    { \skip_horizontal:N -\tabcolsep }
2078                    { \skip_horizontal:N -\arraycolsep }
2079                  \@@_use_arraybox_with_notes_c:
2080                  \bool_if:NTF \l_@@_tabular_bool
2081                    { \skip_horizontal:N -\tabcolsep }
2082                    { \skip_horizontal:N -\arraycolsep }
2083                }
```

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

```
2084              \skip_vertical:N -\l_tmpb_dim
2085              \skip_vertical:N \arrayrulewidth
2086            }
2087          \exp_after:wN \right \g_@@_right_delim_tl
2088          \c_math_toggle_token
2089        }
```

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.

```
2090        \bool_if:NTF \l_@@_delimiters_max_width_bool
2091          {
2092            \@@_put_box_in_flow_bis:nn
2093              \g_@@_left_delim_tl
2094              \g_@@_right_delim_tl
2095          }
2096          \@@_put_box_in_flow:
2097      }
```

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.).

```
2098      \bool_if:NT \g_@@_last_col_found_bool
2099        { \skip_horizontal:N \g_@@_width_last_col_dim }
2100      \bool_if:NT \l_@@_preamble_bool
2101        {
2102          \int_compare:nNnT \c@jCol < \g_@@_static_num_of_col_int
2103            { \@@_warning_gredirect_none:n  { columns~not~used } }
```

---

[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).

```
2104        }
2105      \@@_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.

```
2106      \egroup
```

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

```
2107      \iow_now:Nn \@mainaux { \ExplSyntaxOn }
2108      \iow_now:Nn \@mainaux { \char_set_catcode_space:n { 32 }  }
2109      \iow_now:Nx \@mainaux
2110        {
2111          \tl_gset:cn { c_@@_ \int_use:N \g_@@_env_int _ tl }
2112            { \exp_not:o \g_@@_aux_tl }
2113        }
2114      \iow_now:Nn \@mainaux { \ExplSyntaxOff }


2115      \bool_if:NT \g_@@_footnote_bool \endsavenotes
2116    }
```

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.

```
2117 \cs_new_protected:Npn \@@_transform_preamble:
2118   {
2119     \@@_transform_preamble_i:
2120     \@@_transform_preamble_ii:
2121   }
2122 \cs_new_protected:Npn \@@_transform_preamble_i:
2123   {
2124     \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`).

```
2125     \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.

```
2126     \bool_gset_false:N \g_tmpb_bool
```

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

```
2127     \tl_gclear_new:N \g_@@_pre_cell_tl
```

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

```
2128     \int_zero:N \l_tmpa_int
2129     \tl_gclear:N \g_@@_array_preamble_tl
2130     \tl_if_eq:NNTF \l_@@_vlines_clist \c_@@_all_tl
2131       {
2132         \tl_gset:Nn \g_@@_array_preamble_tl
2133           { ! { \skip_horizontal:N \arrayrulewidth } } }
2134       }
2135       {
2136         \clist_if_in:NnT \l_@@_vlines_clist 1
2137           {
2138             \tl_gset:Nn \g_@@_array_preamble_tl
```

```
2139                { ! { \skip_horizontal:N \arrayrulewidth } }
2140            }
2141        }
```

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

```
2142        \exp_last_unbraced:NV \@@_rec_preamble:n \g_@@_user_preamble_tl \stop
2143        \int_gset_eq:NN \g_@@_static_num_of_col_int \c@jCol

2144        \@@_replace_columncolor:
2145    }


2146 \hook_gput_code:nnn { begindocument } { . }
2147    {
2148        \IfPackageLoadedTF { colortbl }
2149            {
```

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).

```
2150                \regex_const:Nn \c_@@_columncolor_regex { \c { columncolor } }
2151                \cs_new_protected:Npn \@@_replace_columncolor:
2152                    {
2153                        \regex_replace_all:NnN
2154                            \c_@@_columncolor_regex
2155                            { \c { @@_columncolor_preamble } }
2156                            \g_@@_array_preamble_tl
2157                    }
2158            }
2159            {
2160                \cs_new_protected:Npn \@@_replace_columncolor:
2161                    { \cs_set_eq:NN \columncolor \@@_columncolor_preamble }
2162            }
2163    }


2164 \cs_new_protected:Npn \@@_transform_preamble_ii:
2165    {
```

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

```
2166        \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2167            {
2168                \tl_if_eq:NNF \g_@@_right_delim_tl \c_@@_dot_tl
2169                    { \bool_gset_true:N \g_@@_delims_bool }
2170            }
2171            { \bool_gset_true:N \g_@@_delims_bool }
```

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

```
2172        \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).

```
2173        \int_if_zero:nTF \l_@@_first_col_int
2174            { \tl_gput_left:No \g_@@_array_preamble_tl \c_@@_preamble_first_col_tl }
2175            {
2176                \bool_if:NF \g_@@_delims_bool
2177                    {
2178                        \bool_if:NF \l_@@_tabular_bool
2179                            {
2180                                \tl_if_empty:NT \l_@@_vlines_clist
2181                                    {
```

```
2182                      \bool_if:NF \l_@@_exterior_arraycolsep_bool
2183                        { \tl_gput_left:Nn \g_@@_array_preamble_tl { @ { } } } }
2184                  }
2185              }
2186          }
2187      }
2188    \int_compare:nNnTF \l_@@_last_col_int > { -1 }
2189      { \tl_gput_right:No \g_@@_array_preamble_tl \c_@@_preamble_last_col_tl }
2190      {
2191        \bool_if:NF \g_@@_delims_bool
2192          {
2193            \bool_if:NF \l_@@_tabular_bool
2194              {
2195                \tl_if_empty:NT \l_@@_vlines_clist
2196                  {
2197                    \bool_if:NF \l_@@_exterior_arraycolsep_bool
2198                      { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { } } } }
2199                  }
2200              }
2201          }
2202      }
```

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).

```
2203      \dim_compare:nNnT \l_@@_tabular_width_dim = \c_zero_dim
2204        {
2205          \tl_gput_right:Nn \g_@@_array_preamble_tl
2206            { > { \@@_error_too_much_cols: } l }
2207        }
2208  }
```

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.

```
2209 \cs_new_protected:Npn \@@_rec_preamble:n #1
2210   {
```

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]

```
2211      \cs_if_exist:cTF { @@ _ \token_to_str:N #1 }
2212        { \use:c { @@ _ \token_to_str:N #1 } { #1 } }
2213        {
```

Now, the columns defined by \newcolumntype of array.

```
2214          \cs_if_exist:cTF { NC @ find @ #1 }
2215            {
2216              \tl_set_eq:Nc \l_tmpb_tl { NC @ rewrite @ #1 }
2217              \exp_last_unbraced:NV \@@_rec_preamble:n \l_tmpb_tl
2218            }
2219            {
2220              \tl_if_eq:nnT { #1 } { S }
2221                { \@@_fatal:n { unknown~column~type-S } }
2222                { \@@_fatal:nn { unknown~column~type } { #1 } }
2223            }
2224        }
2225  }
```

---

[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 c, l and r

```
2226  \cs_new:Npn \@@_c #1
2227    {
2228      \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2229      \tl_gclear:N \g_@@_pre_cell_tl
2230      \tl_gput_right:Nn \g_@@_array_preamble_tl
2231        { > \@@_cell_begin:w c < \@@_cell_end: }
```

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

```
2232      \int_gincr:N \c@jCol
2233      \@@_rec_preamble_after_col:n
2234    }
2235  \cs_new:Npn \@@_l #1
2236    {
2237      \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2238      \tl_gclear:N \g_@@_pre_cell_tl
2239      \tl_gput_right:Nn \g_@@_array_preamble_tl
2240        {
2241          > { \@@_cell_begin:w \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_l_tl }
2242          l
2243          < \@@_cell_end:
2244        }
2245      \int_gincr:N \c@jCol
2246      \@@_rec_preamble_after_col:n
2247    }
2248  \cs_new:Npn \@@_r #1
2249    {
2250      \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2251      \tl_gclear:N \g_@@_pre_cell_tl
2252      \tl_gput_right:Nn \g_@@_array_preamble_tl
2253        {
2254          > { \@@_cell_begin:w \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_r_tl }
2255          r
2256          < \@@_cell_end:
2257        }
2258      \int_gincr:N \c@jCol
2259      \@@_rec_preamble_after_col:n
2260    }
```

For ! and @

```
2261  \cs_new:cpn { @@ _ \token_to_str:N ! } #1 #2
2262    {
2263      \tl_gput_right:Nn \g_@@_array_preamble_tl { #1 { #2 } }
2264      \@@_rec_preamble:n
2265    }
2266  \cs_set_eq:cc { @@ _ \token_to_str:N @ } { @@ _ \token_to_str:N ! }
```

For |

```
2267  \cs_new:cpn { @@ _ | } #1
2268    {
```

\l_tmpa_int is the number of successive occurrences of |

```
2269      \int_incr:N \l_tmpa_int
2270      \@@_make_preamble_i_i:n
2271    }
2272  \cs_new_protected:Npn \@@_make_preamble_i_i:n #1
2273    {
2274    \str_if_eq:nnTF { #1 } |
2275      { \use:c { @@ _ | } | }
2276      { \@@_make_preamble_i_ii:nn { } #1 }
2277    }
```

```
2278  \cs_new_protected:Npn \@@_make_preamble_i_ii:nn #1 #2
2279    {
2280      \str_if_eq:nnTF { #2 } [
2281        { \@@_make_preamble_i_ii:nw { #1 } [ }
2282        { \@@_make_preamble_i_iii:nn { #2 } { #1 } }
2283    }
2284  \cs_new_protected:Npn \@@_make_preamble_i_ii:nw #1 [ #2 ]
2285    { \@@_make_preamble_i_ii:nn { #1 , #2 } }
2286  \cs_new_protected:Npn \@@_make_preamble_i_iii:nn #1 #2
2287    {
2288      \@@_compute_rule_width:n { multiplicity = \l_tmpa_int , #2 }
2289      \tl_gput_right:Nx \g_@@_array_preamble_tl
2290        {
```

Here, the command \dim_eval:n is mandatory.

```
2291          \exp_not:N ! { \skip_horizontal:n { \dim_eval:n { \l_@@_rule_width_dim } } } }
2292        }
2293      \tl_gput_right:Nx \g_@@_pre_code_after_tl
2294        {
2295          \@@_vline:n
2296            {
2297              position = \int_eval:n { \c@jCol + 1 } ,
2298              multiplicity = \int_use:N \l_tmpa_int ,
2299              total-width = \dim_use:N \l_@@_rule_width_dim ,
2300              #2
2301            }
```

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.

```
2302        }
2303      \int_zero:N \l_tmpa_int
2304      \str_if_eq:nnT { #1 } { \stop } { \bool_gset_true:N \g_tmpb_bool }
2305      \@@_rec_preamble:n #1
2306    }


2307  \cs_new:cpn { @@ _ > } #1 #2
2308    {
2309      \tl_gput_right:Nn \g_@@_pre_cell_tl { > { #2 } }
2310      \@@_rec_preamble:n
2311    }
2312  \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.

```
2313  \keys_define:nn { nicematrix / p-column }
2314    {
2315      r .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_r_str ,
2316      r .value_forbidden:n = true ,
2317      c .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_c_str ,
2318      c .value_forbidden:n = true ,
2319      l .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_l_str ,
2320      l .value_forbidden:n = true ,
2321      R .code:n =
2322        \IfPackageLoadedTF { ragged2e }
2323          { \str_set_eq:NN \l_@@_hpos_col_str \c_@@_R_str }
2324          {
2325            \@@_error_or_warning:n { ragged2e~not~loaded }
2326            \str_set_eq:NN \l_@@_hpos_col_str \c_@@_r_str
2327          } ,
2328      R .value_forbidden:n = true ,
2329      L .code:n =
```

```
2330        \IfPackageLoadedTF { ragged2e }
2331          { \str_set_eq:NN \l_@@_hpos_col_str \c_@@_L_stsr }
2332          {
2333            \@@_error_or_warning:n { ragged2e~not~loaded }
2334            \str_set_eq:NN \l_@@_hpos_col_str \c_@@_l_str
2335          } ,
2336      L .value_forbidden:n = true ,
2337      C .code:n =
2338        \IfPackageLoadedTF { ragged2e }
2339          { \str_set_eq:NN \l_@@_hpos_col_str \c_@@_C_str }
2340          {
2341            \@@_error_or_warning:n { ragged2e~not~loaded }
2342            \str_set_eq:NN \l_@@_hpos_col_str \c_@@_c_str
2343          } ,
2344      C .value_forbidden:n = true ,
2345      S .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_si_str ,
2346      S .value_forbidden:n = true ,
2347      p .code:n = \str_set:Nn \l_@@_vpos_col_str { p } ,
2348      p .value_forbidden:n = true ,
2349      t .meta:n = p ,
2350      m .code:n = \str_set:Nn \l_@@_vpos_col_str { m } ,
2351      m .value_forbidden:n = true ,
2352      b .code:n = \str_set:Nn \l_@@_vpos_col_str { b } ,
2353      b .value_forbidden:n = true ,
2354    }
```

For p but also b and m.

```
2355 \cs_new:Npn \@@_p #1
2356   {
2357     \str_set:Nn \l_@@_vpos_col_str { #1 }
```

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

```
2358     \@@_make_preamble_ii_i:n
2359   }
2360 \cs_set_eq:NN \@@_b \@@_p
2361 \cs_set_eq:NN \@@_m \@@_p
2362 \cs_new_protected:Npn \@@_make_preamble_ii_i:n #1
2363   {
2364     \str_if_eq:nnTF { #1 } { [ }
2365       { \@@_make_preamble_ii_ii:w [ }
2366       { \@@_make_preamble_ii_ii:w [ ] { #1 } }
2367   }
2368 \cs_new_protected:Npn \@@_make_preamble_ii_ii:w [ #1 ]
2369   { \@@_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.

```
2370 \cs_new_protected:Npn \@@_make_preamble_ii_iii:nn #1 #2
2371   {
```

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).

```
2372     \str_set_eq:NN \l_@@_hpos_col_str \c_@@_j_str
2373     \@@_keys_p_column:n { #1 }
2374     \@@_make_preamble_ii_iv:nnn { #2 } { minipage } { }
2375   }
2376 \cs_new_protected:Npn \@@_keys_p_column:n #1
2377   { \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.

```
2378 \cs_new_protected:Npn \@@_make_preamble_ii_iv:nnn #1 #2 #3
```

```
2379    {
2380      \use:e
2381        {
2382          \@@_make_preamble_ii_v:nnnnnnnn
2383            { \str_if_eq:onTF \l_@@_vpos_col_str { p } { t } { b } }
2384            { \dim_eval:n { #1 } }
2385            {
```

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.

```
2386              \str_if_eq:NNTF \l_@@_hpos_col_str \c_@@_j_str
2387                { \tl_clear:N \exp_not:N \l_@@_hpos_cell_tl }
2388                {
2389                  \cs_set_nopar:Npn \exp_not:N \l_@@_hpos_cell_tl
2390                    { \str_lowercase:o \l_@@_hpos_col_str }
2391                }
2392              \str_case:on \l_@@_hpos_col_str
2393                {
2394                  c { \exp_not:N \centering }
2395                  l { \exp_not:N \raggedright }
2396                  r { \exp_not:N \raggedleft }
2397                  C { \exp_not:N \Centering }
2398                  L { \exp_not:N \RaggedRight }
2399                  R { \exp_not:N \RaggedLeft }
2400                }
2401              #3
2402            }
2403            { \str_if_eq:onT \l_@@_vpos_col_str { m } \@@_center_cell_box: }
2404            { \str_if_eq:onT \l_@@_hpos_col_str { si } \siunitx_cell_begin:w }
2405            { \str_if_eq:onT \l_@@_hpos_col_str { si } \siunitx_cell_end: }
2406            { #2 }
2407            {
2408              \str_case:onF \l_@@_hpos_col_str
2409                {
2410                  { j } { c }
2411                  { si } { c }
2412                }
```

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

```
2413                { \str_lowercase:o \l_@@_hpos_col_str }
2414            }
2415        }
```

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

```
2416      \int_gincr:N \c@jCol
2417      \@@_rec_preamble_after_col:n
2418  }
```

#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*.

```
2419  \cs_new_protected:Npn \@@_make_preamble_ii_v:nnnnnnnn #1 #2 #3 #4 #5 #6 #7 #8
2420    {
2421      \tl_if_eq:NNTF \l_@@_hpos_col_str \c_@@_si_str
2422        { \tl_gput_right:Nn \g_@@_array_preamble_tl { > { \@@_test_if_empty_for_S: } } }
2423        { \tl_gput_right:Nn \g_@@_array_preamble_tl { > { \@@_test_if_empty: } } }
2424      \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2425      \tl_gclear:N \g_@@_pre_cell_tl
2426      \tl_gput_right:Nn \g_@@_array_preamble_tl
2427        {
2428          > {
```

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.

```
2429            \dim_set:Nn \l_@@_col_width_dim { #2 }
2430            \bool_if:NT \c_@@_testphase_table_bool
2431              { \tag_struct_begin:n { tag = Div } }
2432            \@@_cell_begin:w
```

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

```
2433            \use:c { #7 } [ #1 ] { #2 }
```

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

```
2434            \everypar
2435              {
2436                \vrule height \box_ht:N \@arstrutbox width \c_zero_dim
2437                \everypar { }
2438              }
2439            \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.).

```
2440            #3
```

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

```
2441            \g_@@_row_style_tl
2442            \arraybackslash
2443            #5
2444          }
2445          #8
2446          < {
2447            #6
```

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

```
2448            \@finalstrut \@arstrutbox
2449            \use:c { end #7 }
```

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

```
2450            #4
2451            \@@_cell_end:
2452            \bool_if:NT \c_@@_testphase_table_bool \tag_struct_end:
2453          }
2454        }
2455    }


2456  \str_new:N \c_@@_ignorespaces_str
2457  \str_set:Nx \c_@@_ignorespaces_str { \ignorespaces }
2458  \str_remove_all:Nn \c_@@_ignorespaces_str { ~ }
```

In order to test whether a cell is empty, we test whether it begins by `\ignorespaces\unskip`. However, in some circumstancies, 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...

```
2459  \cs_new_protected:Npn \@@_test_if_empty: { \peek_after:Nw \@@_test_if_empty_i:  }
```

```
2460  \cs_new_protected:Npn \@@_test_if_empty_i:
2461    {
2462      \str_set:Nx \l_tmpa_str { \token_to_meaning:N \l_peek_token }
2463      \str_if_eq:NNT \l_tmpa_str \c_@@_ignorespaces_str
2464        { \@@_test_if_empty:w }
2465    }
2466  \cs_new_protected:Npn \@@_test_if_empty:w \ignorespaces
2467    {
2468      \peek_meaning:NT \unskip
2469        {
2470          \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2471            {
2472              \box_set_wd:Nn \l_@@_cell_box \c_zero_dim
2473              \skip_horizontal:N \l_@@_col_width_dim
2474            }
2475        }
2476    }
2477  \cs_new_protected:Npn \@@_test_if_empty_for_S:
2478    {
2479      \peek_meaning:NT \__siunitx_table_skip:n
2480        {
2481          \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2482            { \box_set_wd:Nn \l_@@_cell_box \c_zero_dim }
2483        }
2484    }
```

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.

```
2485  \cs_new_protected:Npn \@@_center_cell_box:
2486    {
```

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

```
2487      \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2488        {
2489          \int_compare:nNnT
2490            { \box_ht:N \l_@@_cell_box }
2491            >
```

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).

```
2492            { \box_ht:N \strutbox }
2493            {
2494              \hbox_set:Nn \l_@@_cell_box
2495                {
2496                  \box_move_down:nn
2497                    {
2498                      ( \box_ht:N \l_@@_cell_box - \box_ht:N \@arstrutbox
2499                      + \baselineskip ) / 2
2500                    }
2501                    { \box_use:N \l_@@_cell_box }
2502                }
2503            }
2504        }
2505    }
```

For V (similar to the V of varwidth).

```
2506  \cs_new:Npn \@@_V #1 #2
```

```
2507     {
2508       \str_if_eq:nnTF { #2 } { [ }
2509         { \@@_make_preamble_V_i:w [ }
2510         { \@@_make_preamble_V_i:w [ ] { #2 } }
2511     }
2512   \cs_new_protected:Npn \@@_make_preamble_V_i:w [ #1 ]
2513     { \@@_make_preamble_V_ii:nn { #1 } }
2514   \cs_new_protected:Npn \@@_make_preamble_V_ii:nn #1 #2
2515     {
2516       \str_set:Nn \l_@@_vpos_col_str { p }
2517       \str_set_eq:NN \l_@@_hpos_col_str \c_@@_j_str
2518       \@@_keys_p_column:n { #1 }
2519       \IfPackageLoadedTF { varwidth }
2520         { \@@_make_preamble_ii_iv:nnn { #2 } { varwidth } { } }
2521         {
2522           \@@_error_or_warning:n { varwidth~not~loaded }
2523           \@@_make_preamble_ii_iv:nnn { #2 } { minipage } { }
2524         }
2525     }
```

For `w` and `W`

```
2526   \cs_new:Npn \@@_w { \@@_make_preamble_w:nnnn { } }
2527   \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.

```
2528   \cs_new_protected:Npn \@@_make_preamble_w:nnnn #1 #2 #3 #4
2529     {
2530       \str_if_eq:nnTF { #3 } { s }
2531         { \@@_make_preamble_w_i:nnnn { #1 } { #4 } }
2532         { \@@_make_preamble_w_ii:nnnn { #1 } { #2 } { #3 } { #4 } }
2533     }
```

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.

```
2534   \cs_new_protected:Npn \@@_make_preamble_w_i:nnnn #1 #2
2535     {
2536       \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2537       \tl_gclear:N \g_@@_pre_cell_tl
2538       \tl_gput_right:Nn \g_@@_array_preamble_tl
2539         {
2540           > {
2541               \dim_set:Nn \l_@@_col_width_dim { #2 }
2542               \@@_cell_begin:w
2543               \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_c_tl
2544             }
2545           c
2546           < {
2547               \@@_cell_end_for_w_s:
2548               #1
2549               \@@_adjust_size_box:
2550               \box_use_drop:N \l_@@_cell_box
2551             }
2552         }
2553       \int_gincr:N \c@jCol
2554       \@@_rec_preamble_after_col:n
2555     }
```

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

```
2556 \cs_new_protected:Npn \@@_make_preamble_w_ii:nnnn #1 #2 #3 #4
2557   {
2558     \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2559     \tl_gclear:N \g_@@_pre_cell_tl
2560     \tl_gput_right:Nn \g_@@_array_preamble_tl
2561       {
2562         > {
```

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.

```
2563             \dim_set:Nn \l_@@_col_width_dim { #4 }
2564             \hbox_set:Nw \l_@@_cell_box
2565             \@@_cell_begin:w
2566             \cs_set_nopar:Npn \l_@@_hpos_cell_tl { #3 }
2567           }
2568         c
2569         < {
2570             \@@_cell_end:
2571             \hbox_set_end:
2572             #1
2573             \@@_adjust_size_box:
2574             \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2575           }
2576       }
```

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

```
2577     \int_gincr:N \c@jCol
2578     \@@_rec_preamble_after_col:n
2579   }


2580 \cs_new_protected:Npn \@@_special_W:
2581   {
2582     \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \l_@@_col_width_dim
2583       { \@@_warning:n { W~warning } }
2584   }
```

For S (of siunitx).

```
2585 \cs_new:Npn \@@_S #1 #2
2586   {
2587     \str_if_eq:nnTF { #2 } { [ }
2588       { \@@_make_preamble_S:w [ }
2589       { \@@_make_preamble_S:w [ ] { #2 } }
2590   }
2591 \cs_new_protected:Npn \@@_make_preamble_S:w [ #1 ]
2592   { \@@_make_preamble_S_i:n { #1 } }
2593 \cs_new_protected:Npn \@@_make_preamble_S_i:n #1
2594   {
2595     \IfPackageLoadedTF { siunitx }
2596       {
2597         \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
2598         \tl_gclear:N \g_@@_pre_cell_tl
2599         \tl_gput_right:Nn \g_@@_array_preamble_tl
2600           {
2601             > {
2602                 \@@_cell_begin:w
2603                 \keys_set:nn { siunitx } { #1 }
2604                 \siunitx_cell_begin:w
2605               }
2606             c
2607             < { \siunitx_cell_end: \@@_cell_end: }
2608           }
```

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

```
2609              \int_gincr:N \c@jCol
2610              \@@_rec_preamble_after_col:n
2611          }
2612          { \@@_fatal:n { siunitx~not~loaded } }
2613      }
```

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

```
2614  \cs_new:cpn { @@ _ \token_to_str:N ( } #1 #2
2615      {
2616          \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.

```
2617          \int_if_zero:nTF \c@jCol
2618              {
2619                  \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
2620                      {
```

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

```
2621                          \tl_gset:Nn \g_@@_left_delim_tl { #1 }
2622                          \tl_gset_eq:NN \g_@@_right_delim_tl \c_@@_dot_tl
2623                          \@@_rec_preamble:n #2
2624                      }
2625                      {
2626                          \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2627                          \@@_make_preamble_iv:nn { #1 } { #2 }
2628                      }
2629              }
2630              { \@@_make_preamble_iv:nn { #1 } { #2 } }
2631      }
2632  \cs_set_eq:cc { @@ _ \token_to_str:N [ } { @@ _ \token_to_str:N ( }
2633  \cs_set_eq:cc { @@ _ \token_to_str:N \{ } { @@ _ \token_to_str:N ( }

2634  \cs_new_protected:Npn \@@_make_preamble_iv:nn #1 #2
2635      {
2636          \tl_gput_right:Nx \g_@@_pre_code_after_tl
2637              { \@@_delimiter:nnn #1 { \int_eval:n { \c@jCol + 1 } } \c_true_bool }
2638          \tl_if_in:nnTF { ( [ \{ ) ] \} \left \right } { #2 }
2639              {
2640                  \@@_error:nn { delimiter~after~opening } { #2 }
2641                  \@@_rec_preamble:n
2642              }
2643              { \@@_rec_preamble:n #2 }
2644      }
```

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

```
2645  \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}`).

```
2646  \cs_new:cpn { @@ _ \token_to_str:N ) } #1 #2
2647      {
2648          \bool_if:NT \l_@@_small_bool { \@@_fatal:n { Delimiter~with~small } }
2649          \tl_if_in:nnTF { ) ] \} } { #2 }
2650              { \@@_make_preamble_v:nnn #1 #2 }
2651              {
2652                  \tl_if_eq:nnTF { \stop } { #2 }
2653                      {
2654                          \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2655                              { \tl_gset:Nn \g_@@_right_delim_tl { #1 } }
2656                              {
```

```
2657                \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2658                \tl_gput_right:Nx \g_@@_pre_code_after_tl
2659                  { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2660                \@@_rec_preamble:n #2
2661              }
2662          }
2663          {
2664            \tl_if_in:nnT { ( [ \{ \left } { #2 }
2665              { \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } } }
2666            \tl_gput_right:Nx \g_@@_pre_code_after_tl
2667              { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2668            \@@_rec_preamble:n #2
2669          }
2670        }
2671    }
2672  \cs_set_eq:cc { @@ _ \token_to_str:N ] } { @@ _ \token_to_str:N ) }
2673  \cs_set_eq:cc { @@ _ \token_to_str:N \} } { @@ _ \token_to_str:N ) }
2674  \cs_new_protected:Npn \@@_make_preamble_v:nnn #1 #2 #3
2675    {
2676      \tl_if_eq:nnTF { \stop } { #3 }
2677        {
2678          \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
2679            {
2680              \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2681              \tl_gput_right:Nx \g_@@_pre_code_after_tl
2682                { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2683              \tl_gset:Nn \g_@@_right_delim_tl { #2 }
2684            }
2685            {
2686              \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
2687              \tl_gput_right:Nx \g_@@_pre_code_after_tl
2688                { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2689              \@@_error:nn { double~closing~delimiter } { #2 }
2690            }
2691        }
2692        {
2693          \tl_gput_right:Nx \g_@@_pre_code_after_tl
2694            { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2695          \@@_error:nn { double~closing~delimiter } { #2 }
2696          \@@_rec_preamble:n #3
2697        }
2698    }

2699  \cs_new:cpn { @@ _ \token_to_str:N \right } #1
2700      { \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 @{...}.

```
2701  \cs_new_protected:Npn \@@_rec_preamble_after_col:n #1
2702    {
2703      \str_if_eq:nnTF { #1 } { < }
2704        \@@_rec_preamble_after_col_i:n
2705        {
2706          \str_if_eq:nnTF { #1 } { @ }
2707            \@@_rec_preamble_after_col_ii:n
2708            {
2709              \tl_if_eq:NNTF \l_@@_vlines_clist \c_@@_all_tl
2710                {
2711                  \tl_gput_right:Nn \g_@@_array_preamble_tl
2712                    { ! { \skip_horizontal:N \arrayrulewidth } }
2713                }
```

```
2714              {
2715                \exp_args:NNe
2716                \clist_if_in:NnT \l_@@_vlines_clist { \int_eval:n { \c@jCol + 1 } }
2717                  {
2718                    \tl_gput_right:Nn \g_@@_array_preamble_tl
2719                      { ! { \skip_horizontal:N \arrayrulewidth } }
2720                  }
2721              }
2722            \@@_rec_preamble:n { #1 }
2723          }
2724        }
2725    }
2726  \cs_new_protected:Npn \@@_rec_preamble_after_col_i:n #1
2727    {
2728      \tl_gput_right:Nn \g_@@_array_preamble_tl { < { #1 } }
2729      \@@_rec_preamble_after_col:n
2730    }
```

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.

```
2731  \cs_new_protected:Npn \@@_rec_preamble_after_col_ii:n #1
2732    {
2733      \tl_if_eq:NNTF \l_@@_vlines_clist \c_@@_all_tl
2734        {
2735          \tl_gput_right:Nn \g_@@_array_preamble_tl
2736            { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2737        }
2738        {
2739          \exp_args:NNe
2740          \clist_if_in:NnTF \l_@@_vlines_clist { \int_eval:n { \c@jCol + 1 } }
2741            {
2742              \tl_gput_right:Nn \g_@@_array_preamble_tl
2743                { @ { #1 \skip_horizontal:N \arrayrulewidth } }
2744            }
2745            { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { #1 } } }
2746        }
2747      \@@_rec_preamble:n
2748    }
```

```
2749  \cs_new:cpn { @@ _ * } #1 #2 #3
2750    {
2751      \tl_clear:N \l_tmpa_tl
2752      \int_step_inline:nn { #2 } { \tl_put_right:Nn \l_tmpa_tl { #3 } }
2753      \exp_last_unbraced:No \@@_rec_preamble:n \l_tmpa_tl
2754    }
```

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.

```
2755  \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.

```
2756  \cs_new:Npn \@@_X #1 #2
2757    {
2758      \str_if_eq:nnTF { #2 } { [ }
2759        { \@@_make_preamble_X:w [ }
2760        { \@@_make_preamble_X:w [ ] #2 }
2761    }
2762  \cs_new_protected:Npn \@@_make_preamble_X:w [ #1 ]
2763    { \@@_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`).

```
2764 \keys_define:nn { nicematrix / X-column }
2765   { 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.

```
2766 \cs_new_protected:Npn \@@_make_preamble_X_i:n #1
2767   {
```

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).

```
2768     \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).

```
2769     \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`.

```
2770     \int_zero_new:N \l_@@_weight_int
2771     \int_set_eq:NN \l_@@_weight_int \c_one_int
2772     \@@_keys_p_column:n { #1 }
```

The unknown keys are put in `\l_tmpa_tl`

```
2773     \keys_set:no { nicematrix / X-column } \l_tmpa_tl
2774     \int_compare:nNnT \l_@@_weight_int < \c_zero_int
2775       {
2776         \@@_error_or_warning:n { negative~weight }
2777         \int_set:Nn \l_@@_weight_int { - \l_@@_weight_int }
2778       }
2779     \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).

```
2780     \bool_if:NTF \l_@@_X_columns_aux_bool
2781       {
2782         \exp_args:Nne
2783         \@@_make_preamble_ii_iv:nnn
2784           { \l_@@_weight_int \l_@@_X_columns_dim }
2785           { minipage }
2786           { \@@_no_update_width: }
2787       }
2788       {
2789         \tl_gput_right:Nn \g_@@_array_preamble_tl
2790           {
2791             > {
2792               \@@_cell_begin:w
2793               \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`.

```
2794               \NotEmpty
```

The following code will nullify the box of the cell.

```
2795               \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2796                 { \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.

```
2797                    \begin { minipage } { 5 cm } \arraybackslash
2798                }
2799            c
2800            < {
2801                    \end { minipage }
2802                    \@@_cell_end:
2803                }
2804            }
2805        \int_gincr:N \c@jCol
2806        \@@_rec_preamble_after_col:n
2807        }
2808    }

2809 \cs_new_protected:Npn \@@_no_update_width:
2810    {
2811        \tl_gput_right:Nn \g_@@_cell_after_hook_tl
2812            { \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing: }
2813    }
```

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

```
2814 \cs_new_protected:Npn \@@_make_preamble_vlism:n #1
2815    {
2816        \seq_gput_right:Nx \g_@@_cols_vlism_seq
2817            { \int_eval:n { \c@jCol + 1 } }
2818        \tl_gput_right:Nx \g_@@_array_preamble_tl
2819            { \exp_not:N ! { \skip_horizontal:N \arrayrulewidth } }
2820        \@@_rec_preamble:n
2821    }
```

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.

```
2822 \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).

```
2823 \cs_new_protected:cpn { @@ _ \token_to_str:N \hline }
2824    { \@@_fatal:n { Preamble~forgotten } }
2825 \cs_set_eq:cc { @@ _ \token_to_str:N \Hline } { @@ _ \token_to_str:N \hline }
2826 \cs_set_eq:cc { @@ _ \token_to_str:N \toprule } { @@ _ \token_to_str:N \hline }
2827 \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).

```
2828 \cs_new:Npn \@@_multicolumn:nnn #1 #2 #3
2829    {
```

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.

```
2830        \multispan { #1 }
2831        \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing:
2832        \begingroup
2833        \bool_if:NT \c_@@_testphase_table_bool
2834            { \tbl_update_multicolumn_cell_data:n { #1 } }
2835        \cs_set_nopar:Npn \@addamp
2836            { \legacy_if:nTF { @firstamp } { \@firstampfalse } { \@preamerr 5 } }
```

73

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

```
2837        \tl_gclear:N \g_@@_preamble_tl
2838        \@@_make_m_preamble:n #2 \q_stop
```

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

```
2839        \exp_args:No \@mkpream \g_@@_preamble_tl
2840        \@addtopreamble \@empty
2841        \endgroup
2842        \bool_if:NT \c_@@_testphase_table_bool
2843          { \UseTaggingSocket { tbl / colspan } { #1 } } }
```

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

```
2844        \int_compare:nNnT { #1 } > \c_one_int
2845          {
2846            \seq_gput_left:Nx \g_@@_multicolumn_cells_seq
2847              { \int_use:N \c@iRow - \int_eval:n { \c@jCol + 1 } }
2848            \seq_gput_left:Nn \g_@@_multicolumn_sizes_seq { #1 }
2849            \seq_gput_right:Nx \g_@@_pos_of_blocks_seq
2850              {
2851                {
2852                  \int_if_zero:nTF \c@jCol
2853                    { \int_eval:n { \c@iRow + 1 } }
2854                    { \int_use:N \c@iRow }
2855                }
2856                { \int_eval:n { \c@jCol + 1 } }
2857                {
2858                  \int_if_zero:nTF \c@jCol
2859                    { \int_eval:n { \c@iRow + 1 } }
2860                    { \int_use:N \c@iRow }
2861                }
2862                { \int_eval:n { \c@jCol + #1 } }
2863                { } % for the name of the block
2864              }
2865          }
```

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

```
2866        \cs_set_nopar:Npn \@sharp { #3 }
2867        \@arstrut
2868        \@preamble
2869        \null
```

We add some lines.

```
2870        \int_gadd:Nn \c@jCol { #1 - 1 }
2871        \int_compare:nNnT \c@jCol > \g_@@_col_total_int
2872          { \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
2873        \ignorespaces
2874      }
```

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.

```
2875  \cs_new_protected:Npn \@@_make_m_preamble:n #1
2876    {
2877      \str_case:nnF { #1 }
2878        {
2879          c { \@@_make_m_preamble_i:n #1 }
2880          l { \@@_make_m_preamble_i:n #1 }
2881          r { \@@_make_m_preamble_i:n #1 }
2882          > { \@@_make_m_preamble_ii:nn #1 }
2883          ! { \@@_make_m_preamble_ii:nn #1 }
2884          @ { \@@_make_m_preamble_ii:nn #1 }
2885          | { \@@_make_m_preamble_iii:n #1 }
```

```
2886        p { \@@_make_m_preamble_iv:nnn t #1 }
2887        m { \@@_make_m_preamble_iv:nnn c #1 }
2888        b { \@@_make_m_preamble_iv:nnn b #1 }
2889        w { \@@_make_m_preamble_v:nnnn { } #1 }
2890        W { \@@_make_m_preamble_v:nnnn { \@@_special_W: } #1 }
2891        \q_stop { }
2892      }
2893      {
2894        \cs_if_exist:cTF { NC @ find @ #1 }
2895          {
2896            \tl_set_eq:Nc \l_tmpa_tl { NC @ rewrite @ #1 }
2897            \exp_last_unbraced:No \@@_make_m_preamble:n \l_tmpa_tl
2898          }
2899          {
2900            \tl_if_eq:nnT { #1 } { S }
2901              { \@@_fatal:n { unknown~column~type~S } }
2902              { \@@_fatal:nn { unknown~column~type } { #1 } }
2903          }
2904      }
2905  }
```

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

```
2906 \cs_new_protected:Npn \@@_make_m_preamble_i:n #1
2907   {
2908     \tl_gput_right:Nn \g_@@_preamble_tl
2909       {
2910         > { \@@_cell_begin:w \cs_set_nopar:Npn \l_@@_hpos_cell_tl { #1 } }
2911         #1
2912         < \@@_cell_end:
2913       }
```

We test for the presence of a `<`.

```
2914     \@@_make_m_preamble_x:n
2915   }
```

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

```
2916 \cs_new_protected:Npn \@@_make_m_preamble_ii:nn #1 #2
2917   {
2918     \tl_gput_right:Nn \g_@@_preamble_tl { #1 { #2 } }
2919     \@@_make_m_preamble:n
2920   }
```

For `|`

```
2921 \cs_new_protected:Npn \@@_make_m_preamble_iii:n #1
2922   {
2923     \tl_gput_right:Nn \g_@@_preamble_tl { #1 }
2924     \@@_make_m_preamble:n
2925   }
```

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

```
2926 \cs_new_protected:Npn \@@_make_m_preamble_iv:nnn #1 #2 #3
2927   {
2928     \tl_gput_right:Nn \g_@@_preamble_tl
2929       {
2930         > {
2931             \@@_cell_begin:w
2932             \begin { minipage } [ #1 ] { \dim_eval:n { #3 } }
2933             \mode_leave_vertical:
2934             \arraybackslash
2935             \vrule height \box_ht:N \@arstrutbox depth 0 pt width 0 pt
2936         }
2937         c
2938         < {
```

75

```
2939          \vrule height 0 pt depth \box_dp:N \@arstrutbox width 0 pt
2940          \end { minipage }
2941          \@@_cell_end:
2942        }
2943      }
```

We test for the presence of a `<`.

```
2944      \@@_make_m_preamble_x:n
2945    }
```

For `w` and `W`

```
2946  \cs_new_protected:Npn \@@_make_m_preamble_v:nnnn #1 #2 #3 #4
2947    {
2948      \tl_gput_right:Nn \g_@@_preamble_tl
2949        {
2950          > {
2951              \dim_set:Nn \l_@@_col_width_dim { #4 }
2952              \hbox_set:Nw \l_@@_cell_box
2953              \@@_cell_begin:w
2954              \cs_set_nopar:Npn \l_@@_hpos_cell_tl { #3 }
2955          }
2956          c
2957          < {
2958              \@@_cell_end:
2959              \hbox_set_end:
2960              \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
2961              #1
2962              \@@_adjust_size_box:
2963              \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2964          }
2965        }
```

We test for the presence of a `<`.

```
2966      \@@_make_m_preamble_x:n
2967    }
```

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

```
2968  \cs_new_protected:Npn \@@_make_m_preamble_x:n #1
2969    {
2970      \str_if_eq:nnTF { #1 } { < }
2971        \@@_make_m_preamble_ix:n
2972        { \@@_make_m_preamble:n { #1 } }
2973    }
2974  \cs_new_protected:Npn \@@_make_m_preamble_ix:n #1
2975    {
2976      \tl_gput_right:Nn \g_@@_preamble_tl { < { #1 } }
2977      \@@_make_m_preamble_x:n
2978    }
```

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`).

```
2979  \cs_new_protected:Npn \@@_put_box_in_flow:
2980    {
2981      \box_set_ht:Nn \l_tmpa_box { \box_ht:N \l_tmpa_box + \l_tmpa_dim }
2982      \box_set_dp:Nn \l_tmpa_box { \box_dp:N \l_tmpa_box + \l_tmpb_dim }
2983      \tl_if_eq:NNTF \l_@@_baseline_tl \c_@@_c_tl
2984        { \box_use_drop:N \l_tmpa_box }
2985        \@@_put_box_in_flow_i:
2986    }
```

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).

```
2987 \cs_new_protected:Npn \@@_put_box_in_flow_i:
2988   {
2989     \pgfpicture
2990       \@@_qpoint:n { row - 1 }
2991       \dim_gset_eq:NN \g_tmpa_dim \pgf@y
2992       \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
2993       \dim_gadd:Nn \g_tmpa_dim \pgf@y
2994       \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).

```
2995       \tl_if_in:NnTF \l_@@_baseline_tl { line- }
2996         {
2997           \int_set:Nn \l_tmpa_int
2998             {
2999               \str_range:Nnn
3000                 \l_@@_baseline_tl
3001                 6
3002                 { \tl_count:o \l_@@_baseline_tl }
3003             }
3004           \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
3005         }
3006         {
3007           \tl_if_eq:NnTF \l_@@_baseline_tl { t }
3008             { \int_set_eq:NN \l_tmpa_int \c_one_int }
3009             {
3010               \tl_if_eq:NnTF \l_@@_baseline_tl { b }
3011                 { \int_set_eq:NN \l_tmpa_int \c@iRow }
3012                 { \int_set:Nn \l_tmpa_int \l_@@_baseline_tl }
3013             }
3014           \bool_lazy_or:nnT
3015             { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
3016             { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
3017             {
3018               \@@_error:n { bad~value~for~baseline }
3019               \int_set_eq:NN \l_tmpa_int \c_one_int
3020             }
3021           \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }
```

We take into account the position of the mathematical axis.

```
3022           \dim_gsub:Nn \g_tmpa_dim { \fontdimen22 \textfont2 }
3023         }
3024       \dim_gsub:Nn \g_tmpa_dim \pgf@y
```

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

```
3025     \endpgfpicture
3026     \box_move_up:nn \g_tmpa_dim { \box_use_drop:N \l_tmpa_box }
3027     \box_use_drop:N \l_tmpa_box
3028   }
```

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).

```
3029 \cs_new_protected:Npn \@@_use_arraybox_with_notes_c:
3030   {
```

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.

```
3031     \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3032       {
3033         \int_compare:nNnT \c@jCol > \c_one_int
```

```
3034              {
3035                \box_set_wd:Nn \l_@@_the_array_box
3036                  { \box_wd:N \l_@@_the_array_box - \arraycolsep }
3037              }
3038          }
```

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

```
3039        \begin { minipage } [ t ] { \box_wd:N \l_@@_the_array_box }
3040        \bool_if:NT \l_@@_caption_above_bool
3041          {
3042            \tl_if_empty:NF \l_@@_caption_tl
3043              {
3044                \bool_set_false:N \g_@@_caption_finished_bool
3045                \int_gzero:N \c@tabularnote
3046                \@@_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).

```
3047                \int_compare:nNnT \g_@@_notes_caption_int > \c_zero_int
3048                  {
3049                    \tl_gput_right:Nx \g_@@_aux_tl
3050                      {
3051                        \tl_set:Nn \exp_not:N \l_@@_note_in_caption_tl
3052                          { \int_use:N \g_@@_notes_caption_int }
3053                      }
3054                    \int_gzero:N \g_@@_notes_caption_int
3055                  }
3056              }
3057          }
```

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

```
3058        \hbox
3059          {
3060            \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.

```
3061            \@@_create_extra_nodes:
3062            \seq_if_empty:NF \g_@@_blocks_seq \@@_draw_blocks:
3063          }
```

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

```
3064        \bool_lazy_any:nT
3065          {
3066            { ! \seq_if_empty_p:N \g_@@_notes_seq }
3067            { ! \seq_if_empty_p:N \g_@@_notes_in_caption_seq }
3068            { ! \tl_if_empty_p:o \g_@@_tabularnote_tl }
3069          }
3070          \@@_insert_tabularnotes:
3071        \cs_set_eq:NN \tabularnote \@@_tabularnote_error:n
3072        \bool_if:NF \l_@@_caption_above_bool \@@_insert_caption:
3073      \end { minipage }
3074  }


3075 \cs_new_protected:Npn \@@_insert_caption:
3076  {
```

```
3077        \tl_if_empty:NF \l_@@_caption_tl
3078          {
3079            \cs_if_exist:NTF \@captype
3080              { \@@_insert_caption_i: }
3081              { \@@_error:n { caption~outside~float } }
3082          }
3083      }
```

```
3084 \cs_new_protected:Npn \@@_insert_caption_i:
3085    {
3086        \group_begin:
```

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

```
3087        \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.

```
3088        \IfPackageLoadedTF { floatrow }
3089          { \cs_set_eq:NN \@makecaption \FR@makecaption }
3090          { }
3091        \tl_if_empty:NTF \l_@@_short_caption_tl
3092          { \caption }
3093          { \caption [ \l_@@_short_caption_tl ] }
3094          { \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.

```
3095        \bool_if:NF \g_@@_caption_finished_bool
3096          {
3097            \bool_gset_true:N \g_@@_caption_finished_bool
3098            \int_gset_eq:NN \g_@@_notes_caption_int \c@tabularnote
3099            \int_gzero:N \c@tabularnote
3100          }
3101        \tl_if_empty:NF \l_@@_label_tl { \label { \l_@@_label_tl } }
3102        \group_end:
3103    }
```

```
3104 \cs_new_protected:Npn \@@_tabularnote_error:n #1
3105    {
3106        \@@_error_or_warning:n { tabularnote~below~the~tabular }
3107        \@@_gredirect_none:n { tabularnote~below~the~tabular }
3108    }
```

```
3109 \cs_new_protected:Npn \@@_insert_tabularnotes:
3110    {
3111        \seq_gconcat:NNN \g_@@_notes_seq \g_@@_notes_in_caption_seq \g_@@_notes_seq
3112        \int_set:Nn \c@tabularnote { \seq_count:N \g_@@_notes_seq }
3113        \skip_vertical:N 0.65ex
```

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

```
3114        \group_begin:
3115        \l_@@_notes_code_before_tl
3116        \tl_if_empty:NF \g_@@_tabularnote_tl
3117          {
3118            \g_@@_tabularnote_tl \par
3119            \tl_gclear:N \g_@@_tabularnote_tl
3120          }
```

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.

```
3121    \int_compare:nNnT \c@tabularnote > \c_zero_int
3122      {
3123        \bool_if:NTF \l_@@_notes_para_bool
3124          {
3125            \begin { tabularnotes* }
3126              \seq_map_inline:Nn \g_@@_notes_seq
3127                { \@@_one_tabularnote:nn ##1 }
3128            \strut
3129            \end { tabularnotes* }
```

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

```
3130            \par
3131          }
3132          {
3133            \tabularnotes
3134              \seq_map_inline:Nn \g_@@_notes_seq
3135                { \@@_one_tabularnote:nn ##1 }
3136            \strut
3137            \endtabularnotes
3138          }
3139        }
3140      \unskip
3141      \group_end:
3142      \bool_if:NT \l_@@_notes_bottomrule_bool
3143        {
3144          \IfPackageLoadedTF { booktabs }
3145            {
```

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

```
3146              \skip_vertical:N \aboverulesep
```

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

```
3147              { \CT@arc@ \hrule height \heavyrulewidth }
3148            }
3149            { \@@_error_or_warning:n { bottomrule~without~booktabs } }
3150        }
3151      \l_@@_notes_code_after_tl
3152      \seq_gclear:N \g_@@_notes_seq
3153      \seq_gclear:N \g_@@_notes_in_caption_seq
3154      \int_gzero:N \c@tabularnote
3155    }
```

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.

```
3156 \cs_set_protected:Npn \@@_one_tabularnote:nn #1
3157    {
3158      \tl_if_novalue:nTF { #1 }
3159        { \item }
3160        { \item [ \@@_notes_label_in_list:n { #1 } ] }
3161    }
```

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`.

```
3162 \cs_new_protected:Npn \@@_use_arraybox_with_notes_b:
3163    {
3164      \pgfpicture
3165        \@@_qpoint:n { row - 1 }
3166        \dim_gset_eq:NN \g_tmpa_dim \pgf@y
```

```
3167        \@@_qpoint:n { row - \int_use:N \c@iRow - base }
3168        \dim_gsub:Nn \g_tmpa_dim \pgf@y
3169     \endpgfpicture
3170     \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3171     \int_if_zero:nT \l_@@_first_row_int
3172       {
3173          \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3174          \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3175       }
3176     \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3177   }
```

Now, the general case.

```
3178 \cs_new_protected:Npn \@@_use_arraybox_with_notes:
3179    {
```

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

```
3180      \tl_if_eq:NnT \l_@@_baseline_tl { t }
3181        { \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`.

```
3182      \pgfpicture
3183      \@@_qpoint:n { row - 1 }
3184      \dim_gset_eq:NN \g_tmpa_dim \pgf@y
3185      \str_if_in:NnTF \l_@@_baseline_tl { line- }
3186        {
3187          \int_set:Nn \l_tmpa_int
3188            {
3189              \str_range:Nnn
3190                \l_@@_baseline_tl
3191                6
3192                { \tl_count:o \l_@@_baseline_tl }
3193            }
3194          \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
3195        }
3196        {
3197          \int_set:Nn \l_tmpa_int \l_@@_baseline_tl
3198          \bool_lazy_or:nnT
3199            { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
3200            { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
3201            {
3202              \@@_error:n { bad~value~for~baseline }
3203              \int_set:Nn \l_tmpa_int 1
3204            }
3205          \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }
3206        }
3207      \dim_gsub:Nn \g_tmpa_dim \pgf@y
3208      \endpgfpicture
3209      \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
3210      \int_if_zero:nT \l_@@_first_row_int
3211        {
3212          \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
3213          \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
3214        }
3215      \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
3216   }
```

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.

```
3217 \cs_new_protected:Npn \@@_put_box_in_flow_bis:nn #1 #2
3218    {
```

We will compute the real width of both delimiters used.

```
3219       \dim_zero_new:N \l_@@_real_left_delim_dim
3220       \dim_zero_new:N \l_@@_real_right_delim_dim
3221       \hbox_set:Nn \l_tmpb_box
3222         {
3223           \c_math_toggle_token
3224           \left #1
3225           \vcenter
3226             {
3227               \vbox_to_ht:nn
3228                 { \box_ht_plus_dp:N \l_tmpa_box }
3229                 { }
3230             }
3231           \right .
3232           \c_math_toggle_token
3233         }
3234       \dim_set:Nn \l_@@_real_left_delim_dim
3235         { \box_wd:N \l_tmpb_box - \nulldelimiterspace }
3236       \hbox_set:Nn \l_tmpb_box
3237         {
3238           \c_math_toggle_token
3239           \left .
3240           \vbox_to_ht:nn
3241             { \box_ht_plus_dp:N \l_tmpa_box }
3242             { }
3243           \right #2
3244           \c_math_toggle_token
3245         }
3246       \dim_set:Nn \l_@@_real_right_delim_dim
3247         { \box_wd:N \l_tmpb_box - \nulldelimiterspace }
```

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

```
3248       \skip_horizontal:N  \l_@@_left_delim_dim
3249       \skip_horizontal:N -\l_@@_real_left_delim_dim
3250       \@@_put_box_in_flow:
3251       \skip_horizontal:N \l_@@_right_delim_dim
3252       \skip_horizontal:N -\l_@@_real_right_delim_dim
3253     }
```

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).

```
3254 \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).

```
3255   {
3256     \peek_remove_spaces:n
3257       {
3258         \peek_meaning:NTF \end
3259           \@@_analyze_end:Nn
3260           {
3261             \@@_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).

```
3262             \exp_args:No \@@_array: \g_@@_array_preamble_tl
3263           }
3264       }
3265   }
```

```
3266    {
3267      \@@_create_col_nodes:
3268      \endarray
3269    }
```

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

```
3270  \NewDocumentEnvironment { @@-light-syntax } { b }
3271    {
```

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`.

```
3272      \tl_if_empty:nT { #1 } { \@@_fatal:n { empty~environment } }
3273      \tl_map_inline:nn { #1 }
3274        {
3275          \str_if_eq:nnT { ##1 } { & }
3276            { \@@_fatal:n { ampersand~in~light-syntax } }
3277          \str_if_eq:nnT { ##1 } { \\ }
3278            { \@@_fatal:n { double-backslash~in~light-syntax } }
3279        }
```

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`.

```
3280      \@@_light_syntax_i:w #1 \CodeAfter \q_stop
```

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

```
3281    }
```

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.

```
3282    {
3283      \@@_create_col_nodes:
3284      \endarray
3285    }
3286  \cs_new_protected:Npn \@@_light_syntax_i:w #1\CodeAfter #2\q_stop
3287    {
3288      \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).

```
3289      \seq_clear_new:N \l_@@_rows_seq
```

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

```
3290      \tl_set_rescan:Nno \l_@@_end_of_row_tl { } \l_@@_end_of_row_tl
3291      \bool_if:NTF \l_@@_light_syntax_expanded_bool
3292        \seq_set_split:Nee
3293        \seq_set_split:Non
3294        \l_@@_rows_seq \l_@@_end_of_row_tl { #1 }
```

We delete the last row if it is empty.

```
3295      \seq_pop_right:NN \l_@@_rows_seq \l_tmpa_tl
3296      \tl_if_empty:NF \l_tmpa_tl
3297        { \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.

```
3298      \int_compare:nNnT \l_@@_last_row_int = { -1 }
3299        { \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).

```
3300        \tl_build_begin:N \l_@@_new_body_tl
3301        \int_zero_new:N \l_@@_nb_cols_int
```

First, we treat the first row.

```
3302        \seq_pop_left:NN \l_@@_rows_seq \l_tmpa_tl
3303        \@@_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).

```
3304        \seq_map_inline:Nn \l_@@_rows_seq
3305          {
3306            \tl_build_put_right:Nn \l_@@_new_body_tl { \\ }
3307            \@@_line_with_light_syntax:n { ##1 }
3308          }
3309        \tl_build_end:N \l_@@_new_body_tl
3310        \int_compare:nNnT \l_@@_last_col_int = { -1 }
3311          {
3312            \int_set:Nn \l_@@_last_col_int
3313              { \l_@@_nb_cols_int - 1 + \l_@@_first_col_int }
3314          }
```

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.

```
3315        \@@_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).

```
3316        \exp_args:No \@@_array: \g_@@_array_preamble_tl \l_@@_new_body_tl
3317      }
3318 \cs_new_protected:Npn \@@_line_with_light_syntax:n #1
3319    {
3320      \seq_clear_new:N \l_@@_cells_seq
3321      \seq_set_split:Nnn \l_@@_cells_seq { ~ } { #1 }
3322      \int_set:Nn \l_@@_nb_cols_int
3323        {
3324          \int_max:nn
3325            \l_@@_nb_cols_int
3326            { \seq_count:N \l_@@_cells_seq }
3327        }
3328      \seq_pop_left:NN \l_@@_cells_seq \l_tmpa_tl
3329      \exp_args:NNo \tl_build_put_right:Nn \l_@@_new_body_tl \l_tmpa_tl
3330      \seq_map_inline:Nn \l_@@_cells_seq
3331        { \tl_build_put_right:Nn \l_@@_new_body_tl { & ##1 } }
3332    }
3333 \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.

```
3334 \cs_new_protected:Npn \@@_analyze_end:Nn #1 #2
3335    {
3336      \str_if_eq:onT \g_@@_name_env_str { #2 }
3337        { \@@_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.

```
3338      \end { #2 }
3339    }
```

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`).

```
3340 \cs_new:Npn \@@_create_col_nodes:
3341   {
3342     \crcr
3343     \int_if_zero:nT \l_@@_first_col_int
3344       {
3345         \omit
3346         \hbox_overlap_left:n
3347           {
3348             \bool_if:NT \l_@@_code_before_bool
3349               { \pgfsys@markposition { \@@_env: - col - 0 } }
3350             \pgfpicture
3351             \pgfrememberpicturepositiononpagetrue
3352             \pgfcoordinate { \@@_env: - col - 0 } \pgfpointorigin
3353             \str_if_empty:NF \l_@@_name_str
3354               { \pgfnodealias { \l_@@_name_str - col - 0 } { \@@_env: - col - 0 } }
3355             \endpgfpicture
3356             \skip_horizontal:N 2\col@sep
3357             \skip_horizontal:N \g_@@_width_first_col_dim
3358           }
3359         &
3360       }
3361     \omit
```

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

```
3362     \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`).

```
3363     \int_if_zero:nTF \l_@@_first_col_int
3364       {
3365         \bool_if:NT \l_@@_code_before_bool
3366           {
3367             \hbox
3368               {
3369                 \skip_horizontal:N -0.5\arrayrulewidth
3370                 \pgfsys@markposition { \@@_env: - col - 1 }
3371                 \skip_horizontal:N 0.5\arrayrulewidth
3372               }
3373           }
3374         \pgfpicture
3375         \pgfrememberpicturepositiononpagetrue
3376         \pgfcoordinate { \@@_env: - col - 1 }
3377           { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3378         \str_if_empty:NF \l_@@_name_str
3379           { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
3380         \endpgfpicture
3381       }
3382       {
3383         \bool_if:NT \l_@@_code_before_bool
3384           {
3385             \hbox
3386               {
3387                 \skip_horizontal:N 0.5\arrayrulewidth
3388                 \pgfsys@markposition { \@@_env: - col - 1 }
3389                 \skip_horizontal:N -0.5\arrayrulewidth
3390               }
3391           }
3392         \pgfpicture
3393         \pgfrememberpicturepositiononpagetrue
3394         \pgfcoordinate { \@@_env: - col - 1 }
3395           { \pgfpoint { 0.5 \arrayrulewidth } \c_zero_dim }
```

```
3396        \str_if_empty:NF \l_@@_name_str
3397          { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
3398        \endpgfpicture
3399      }
```

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.

```
3400      \skip_gset:Nn \g_tmpa_skip { 0 pt~plus 1 fill }
3401      \bool_if:NF \l_@@_auto_columns_width_bool
3402        { \dim_compare:nNnT \l_@@_columns_width_dim > \c_zero_dim }
3403        {
3404          \bool_lazy_and:nnTF
3405            \l_@@_auto_columns_width_bool
3406            { \bool_not_p:n \l_@@_block_auto_columns_width_bool }
3407            { \skip_gadd:Nn \g_tmpa_skip \g_@@_max_cell_width_dim }
3408            { \skip_gadd:Nn \g_tmpa_skip \l_@@_columns_width_dim }
3409          \skip_gadd:Nn \g_tmpa_skip { 2 \col@sep }
3410        }
3411      \skip_horizontal:N \g_tmpa_skip
3412      \hbox
3413        {
3414          \bool_if:NT \l_@@_code_before_bool
3415            {
3416              \hbox
3417                {
3418                  \skip_horizontal:N -0.5\arrayrulewidth
3419                  \pgfsys@markposition { \@@_env: - col - 2 }
3420                  \skip_horizontal:N 0.5\arrayrulewidth
3421                }
3422            }
3423          \pgfpicture
3424          \pgfrememberpicturepositiononpagetrue
3425          \pgfcoordinate { \@@_env: - col - 2 }
3426            { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3427          \str_if_empty:NF \l_@@_name_str
3428            { \pgfnodealias { \l_@@_name_str - col - 2 } { \@@_env: - col - 2 } }
3429          \endpgfpicture
3430        }
```

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.

```
3431      \int_gset_eq:NN \g_tmpa_int \c_one_int
3432      \bool_if:NTF \g_@@_last_col_found_bool
3433        { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 3 } \c_zero_int } }
3434        { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 2 } \c_zero_int } }
3435        {
3436          &
3437          \omit
3438          \int_gincr:N \g_tmpa_int
```

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

```
3439          \skip_horizontal:N \g_tmpa_skip
3440          \bool_if:NT \l_@@_code_before_bool
3441            {
3442              \hbox
3443                {
3444                  \skip_horizontal:N -0.5\arrayrulewidth
3445                  \pgfsys@markposition
3446                    { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3447                  \skip_horizontal:N 0.5\arrayrulewidth
```

```
3448                    }
3449                }
```
We create the `col` node on the right of the current column.
```
3450            \pgfpicture
3451                \pgfrememberpicturepositiononpagetrue
3452                \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3453                    { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
3454                \str_if_empty:NF \l_@@_name_str
3455                    {
3456                        \pgfnodealias
3457                            { \l_@@_name_str - col - \int_eval:n { \g_tmpa_int + 1 } }
3458                            { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3459                    }
3460            \endpgfpicture
3461        }


3462        &
3463        \omit
```
The two following lines have been added on 2021-12-15 to solve a bug mentionned by Joao Luis Soares by mail.
```
3464        \int_if_zero:nT \g_@@_col_total_int
3465            { \skip_gset:Nn \g_tmpa_skip { 0 pt~plus 1 fill } }
3466        \skip_horizontal:N \g_tmpa_skip
3467        \int_gincr:N \g_tmpa_int
3468        \bool_lazy_any:nF
3469            {
3470                \g_@@_delims_bool
3471                \l_@@_tabular_bool
3472                { ! \clist_if_empty_p:N \l_@@_vlines_clist }
3473                \l_@@_exterior_arraycolsep_bool
3474                \l_@@_bar_at_end_of_pream_bool
3475            }
3476            { \skip_horizontal:N -\col@sep }
3477        \bool_if:NT \l_@@_code_before_bool
3478            {
3479                \hbox
3480                    {
3481                        \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.
```
3482                        \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3483                            { \skip_horizontal:N -\arraycolsep }
3484                        \pgfsys@markposition
3485                            { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3486                        \skip_horizontal:N 0.5\arrayrulewidth
3487                        \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3488                            { \skip_horizontal:N \arraycolsep }
3489                    }
3490            }
3491        \pgfpicture
3492            \pgfrememberpicturepositiononpagetrue
3493            \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3494                {
3495                    \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3496                        {
3497                            \pgfpoint
3498                                { - 0.5 \arrayrulewidth - \arraycolsep }
3499                                \c_zero_dim
3500                        }
3501                        { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
```

```
3502                    }
3503            \str_if_empty:NF \l_@@_name_str
3504              {
3505                \pgfnodealias
3506                  { \l_@@_name_str - col - \int_eval:n { \g_tmpa_int + 1 } }
3507                  { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
3508              }
3509          \endpgfpicture


3510      \bool_if:NT \g_@@_last_col_found_bool
3511        {
3512          \hbox_overlap_right:n
3513            {
3514              \skip_horizontal:N \g_@@_width_last_col_dim
3515              \skip_horizontal:N \col@sep
3516              \bool_if:NT \l_@@_code_before_bool
3517                {
3518                  \pgfsys@markposition
3519                    { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3520                }
3521              \pgfpicture
3522              \pgfrememberpicturepositiononpagetrue
3523              \pgfcoordinate
3524                { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3525                \pgfpointorigin
3526              \str_if_empty:NF \l_@@_name_str
3527                {
3528                  \pgfnodealias
3529                    {
3530                      \l_@@_name_str - col
3531                      - \int_eval:n { \g_@@_col_total_int + 1 }
3532                    }
3533                    { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
3534                }
3535              \endpgfpicture
3536            }
3537        }
3538    % \cr
3539    }
```

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

```
3540  \tl_const:Nn \c_@@_preamble_first_col_tl
3541    {
3542      >
3543        {
```

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).

```
3544          \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
3545          \bool_gset_true:N \g_@@_after_col_zero_bool
3546          \@@_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.

```
3547          \hbox_set:Nw \l_@@_cell_box
3548          \@@_math_toggle:
3549          \@@_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".

```
3550          \int_compare:nNnT \c@iRow > \c_zero_int
3551            {
```

```
3552        \bool_lazy_or:nnT
3553          { \int_compare_p:nNn \l_@@_last_row_int < \c_zero_int }
3554          { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
3555          {
3556            \l_@@_code_for_first_col_tl
3557            \xglobal \colorlet { nicematrix-first-col } { . }
3558          }
3559        }
3560      }
```

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.

```
3561      l
3562      <
3563      {
3564        \@@_math_toggle:
3565        \hbox_set_end:
3566        \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
3567        \@@_adjust_size_box:
3568        \@@_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.

```
3569        \dim_gset:Nn \g_@@_width_first_col_dim
3570          { \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.

```
3571        \hbox_overlap_left:n
3572          {
3573            \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > \c_zero_dim
3574              \@@_node_for_cell:
3575              { \box_use_drop:N \l_@@_cell_box }
3576            \skip_horizontal:N \l_@@_left_delim_dim
3577            \skip_horizontal:N \l_@@_left_margin_dim
3578            \skip_horizontal:N \l_@@_extra_left_margin_dim
3579          }
3580        \bool_gset_false:N \g_@@_empty_cell_bool
3581        \skip_horizontal:N -2\col@sep
3582      }
3583    }
```

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

```
3584 \tl_const:Nn \c_@@_preamble_last_col_tl
3585    {
3586      >
3587      {
3588        \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).

```
3589        \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.

```
3590        \bool_gset_true:N \g_@@_last_col_found_bool
3591        \int_gincr:N \c@jCol
3592        \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.

```
3593        \hbox_set:Nw \l_@@_cell_box
3594          \@@_math_toggle:
3595          \@@_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".

```
3596            \int_compare:nNnT \c@iRow > \c_zero_int
3597              {
3598                \bool_lazy_or:nnT
3599                  { \int_compare_p:nNn \l_@@_last_row_int < \c_zero_int }
3600                  { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
3601                  {
3602                    \l_@@_code_for_last_col_tl
3603                    \xglobal \colorlet { nicematrix-last-col } { . }
3604                  }
3605              }
3606          }
3607      l
3608      <
3609        {
3610          \@@_math_toggle:
3611          \hbox_set_end:
3612          \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
3613          \@@_adjust_size_box:
3614          \@@_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.

```
3615          \dim_gset:Nn \g_@@_width_last_col_dim
3616            { \dim_max:nn \g_@@_width_last_col_dim { \box_wd:N \l_@@_cell_box } }
3617          \skip_horizontal:N -2\col@sep
```

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

```
3618          \hbox_overlap_right:n
3619            {
3620              \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \c_zero_dim
3621                {
3622                  \skip_horizontal:N \l_@@_right_delim_dim
3623                  \skip_horizontal:N \l_@@_right_margin_dim
3624                  \skip_horizontal:N \l_@@_extra_right_margin_dim
3625                  \@@_node_for_cell:
3626                }
3627            }
3628          \bool_gset_false:N \g_@@_empty_cell_bool
3629        }
3630    }
```

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

```
3631 \NewDocumentEnvironment { NiceArray } { }
3632   {
3633     \bool_gset_false:N \g_@@_delims_bool
3634     \str_if_empty:NT \g_@@_name_env_str
3635       { \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).

```
3636     \NiceArrayWithDelims . .
3637   }
3638   { \endNiceArrayWithDelims }
```

We create the variants of the environment {NiceArrayWithDelims}.

```
3639 \cs_new_protected:Npn \@@_def_env:nnn #1 #2 #3
3640   {
3641     \NewDocumentEnvironment { #1 NiceArray } { }
3642       {
```

```
3643          \bool_gset_true:N \g_@@_delims_bool
3644          \str_if_empty:NT \g_@@_name_env_str
3645            { \str_gset:Nn \g_@@_name_env_str { #1 NiceArray } } }
3646          \@@_test_if_math_mode:
3647          \NiceArrayWithDelims #2 #3
3648        }
3649        { \endNiceArrayWithDelims }
3650    }
3651 \@@_def_env:nnn p ( )
3652 \@@_def_env:nnn b [ ]
3653 \@@_def_env:nnn B \{ \}
3654 \@@_def_env:nnn v | |
3655 \@@_def_env:nnn V \| \|
```

14   The environment {NiceMatrix} and its variants

```
3656 \cs_new_protected:Npn \@@_begin_of_NiceMatrix:nn #1 #2
3657    {
3658      \bool_set_false:N \l_@@_preamble_bool
3659      \tl_clear:N \l_tmpa_tl
3660      \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
3661        { \tl_set:Nn \l_tmpa_tl { @ { } } }
3662      \tl_put_right:Nn \l_tmpa_tl
3663        {
3664          *
3665            {
3666              \int_case:nnF \l_@@_last_col_int
3667                {
3668                  { -2 } { \c@MaxMatrixCols }
3669                  { -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).

```
3670                }
3671              { \int_eval:n { \l_@@_last_col_int - 1 } }
3672            }
3673          { #2 }
3674        }
3675      \tl_set:Nn \l_tmpb_tl { \use:c { #1 NiceArray } }
3676      \exp_args:No \l_tmpb_tl \l_tmpa_tl
3677    }
3678 \cs_generate_variant:Nn \@@_begin_of_NiceMatrix:nn { n V }
3679 \clist_map_inline:nn { p , b , B , v , V }
3680    {
3681      \NewDocumentEnvironment { #1 NiceMatrix } { ! O { } }
3682        {
3683          \bool_gset_true:N \g_@@_delims_bool
3684          \str_gset:Nn \g_@@_name_env_str { #1 NiceMatrix }
3685          \int_if_zero:nT \l_@@_last_col_int
3686            {
3687              \bool_set_true:N \l_@@_last_col_without_value_bool
3688              \int_set:Nn \l_@@_last_col_int { -1 }
3689            }
3690          \keys_set:nn { NiceMatrix / NiceMatrix } { ##1 }
3691          \@@_begin_of_NiceMatrix:nV { #1 } \l_@@_columns_type_tl
3692        }
3693        { \use:c { end #1 NiceArray } }
3694    }
```

We define also an environment {NiceMatrix}

```
3695  \NewDocumentEnvironment { NiceMatrix } { ! O { } }
3696    {
3697      \str_gset:Nn \g_@@_name_env_str { NiceMatrix }
3698      \int_if_zero:nT \l_@@_last_col_int
3699        {
3700          \bool_set_true:N \l_@@_last_col_without_value_bool
3701          \int_set:Nn \l_@@_last_col_int { -1 }
3702        }
3703      \keys_set:nn { NiceMatrix / NiceMatrix } { #1 }
3704      \bool_lazy_or:nnT
3705        { \clist_if_empty_p:N \l_@@_vlines_clist }
3706        { \l_@@_except_borders_bool }
3707        { \bool_set_true:N \l_@@_NiceMatrix_without_vlines_bool }
3708      \@@_begin_of_NiceMatrix:nV { } \l_@@_columns_type_tl
3709    }
3710    { \endNiceArray }
```

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

```
3711  \cs_new_protected:Npn \@@_NotEmpty:
3712    { \bool_gset_true:N \g_@@_not_empty_cell_bool }
```

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

```
3713  \NewDocumentEnvironment { NiceTabular } { O { } m ! O { } }
3714    {
```

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`.

```
3715      \dim_compare:nNnT \l_@@_width_dim = \c_zero_dim
3716        { \dim_set_eq:NN \l_@@_width_dim \linewidth }
3717      \str_gset:Nn \g_@@_name_env_str { NiceTabular }
3718      \keys_set:nn { NiceMatrix / NiceTabular } { #1 , #3 }
3719      \tl_if_empty:NF \l_@@_short_caption_tl
3720        {
3721          \tl_if_empty:NT \l_@@_caption_tl
3722            {
3723              \@@_error_or_warning:n { short-caption~without~caption }
3724              \tl_set_eq:NN \l_@@_caption_tl \l_@@_short_caption_tl
3725            }
3726        }
3727      \tl_if_empty:NF \l_@@_label_tl
3728        {
3729          \tl_if_empty:NT \l_@@_caption_tl
3730            { \@@_error_or_warning:n { label~without~caption } }
3731        }
3732      \NewDocumentEnvironment { TabularNote } { b }
3733        {
3734          \bool_if:NTF \l_@@_in_code_after_bool
3735            { \@@_error_or_warning:n { TabularNote~in~CodeAfter } }
3736            {
3737              \tl_if_empty:NF \g_@@_tabularnote_tl
3738                { \tl_gput_right:Nn \g_@@_tabularnote_tl { \par } }
3739              \tl_gput_right:Nn \g_@@_tabularnote_tl { ##1 }
3740            }
3741        }
3742        { }
3743      \@@_settings_for_tabular:
3744      \NiceArray { #2 }
3745    }
3746    {
3747      \endNiceArray
3748      \bool_if:NT \c_@@_testphase_table_bool
```

```
3749        { \UseTaggingSocket { tbl / hmode / end } }
3750    }
3751 \cs_new_protected:Npn \@@_settings_for_tabular:
3752    {
3753      \bool_set_true:N \l_@@_tabular_bool
3754      \cs_set_eq:NN \@@_math_toggle: \prg_do_nothing:
3755      \cs_set_eq:NN \@@_tuning_not_tabular_begin: \prg_do_nothing:
3756      \cs_set_eq:NN \@@_tuning_not_tabular_end: \prg_do_nothing:
3757    }


3758 \NewDocumentEnvironment { NiceTabularX } { m O { } m ! O { } }
3759    {
3760      \str_gset:Nn \g_@@_name_env_str { NiceTabularX }
3761      \dim_zero_new:N \l_@@_width_dim
3762      \dim_set:Nn \l_@@_width_dim { #1 }
3763      \keys_set:nn { NiceMatrix / NiceTabular } { #2 , #4 }
3764      \@@_settings_for_tabular:
3765      \NiceArray { #3 }
3766    }
3767    {
3768      \endNiceArray
3769      \int_if_zero:nT \g_@@_total_X_weight_int
3770        { \@@_error:n { NiceTabularX~without~X } }
3771    }


3772 \NewDocumentEnvironment { NiceTabular* } { m O { } m ! O { } }
3773    {
3774      \str_gset:Nn \g_@@_name_env_str { NiceTabular* }
3775      \dim_set:Nn \l_@@_tabular_width_dim { #1 }
3776      \keys_set:nn { NiceMatrix / NiceTabular } { #2 , #4 }
3777      \@@_settings_for_tabular:
3778      \NiceArray { #3 }
3779    }
3780    { \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).

```
3781 \cs_new_protected:Npn \@@_deal_with_rounded_corners:
3782    {
3783      \bool_lazy_all:nT
3784        {
3785          { \int_compare_p:nNn \l_@@_tab_rounded_corners_dim > \c_zero_dim }
3786          \l_@@_hvlines_bool
3787          { ! \g_@@_delims_bool }
3788          { ! \l_@@_except_borders_bool }
3789        }
3790        {
3791          \bool_set_true:N \l_@@_except_borders_bool
3792          \clist_if_empty:NF \l_@@_corners_clist
3793            { \@@_error:n { hvlines,~rounded-corners~and~corners } }
3794          \tl_gput_right:Nn \g_@@_pre_code_after_tl
3795            {
3796              \@@_stroke_block:nnn
3797                {
3798                  rounded-corners = \dim_use:N \l_@@_tab_rounded_corners_dim ,
3799                  draw = \l_@@_rules_color_tl
```

```
3800                    }
3801                  { 1-1 }
3802                  { \int_use:N \c@iRow - \int_use:N \c@jCol }
3803              }
3804          }
3805    }
```

```
3806 \cs_new_protected:Npn \@@_after_array:
3807    {
```

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.

```
3808      \hook_gremove_code:nn { env / tabular / begin } { nicematrix }
3809      \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.

```
3810      \bool_if:NT \g_@@_last_col_found_bool
3811        { \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`.

```
3812      \bool_if:NT \l_@@_last_col_without_value_bool
3813        { \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.

```
3814      \bool_if:NT \l_@@_last_row_without_value_bool
3815        { \int_set_eq:NN \l_@@_last_row_int \g_@@_row_total_int }
```

```
3816      \tl_gput_right:Nx \g_@@_aux_tl
3817        {
3818          \seq_gset_from_clist:Nn \exp_not:N \g_@@_size_seq
3819            {
3820              \int_use:N \l_@@_first_row_int ,
3821              \int_use:N \c@iRow ,
3822              \int_use:N \g_@@_row_total_int ,
3823              \int_use:N \l_@@_first_col_int ,
3824              \int_use:N \c@jCol ,
3825              \int_use:N \g_@@_col_total_int
3826            }
3827        }
```

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).

```
3828      \seq_if_empty:NF \g_@@_pos_of_blocks_seq
3829        {
3830          \tl_gput_right:Nx \g_@@_aux_tl
3831            {
3832              \seq_gset_from_clist:Nn \exp_not:N \g_@@_pos_of_blocks_seq
3833                { \seq_use:Nnnn \g_@@_pos_of_blocks_seq , , , }
3834            }
3835        }
3836      \seq_if_empty:NF \g_@@_multicolumn_cells_seq
3837        {
3838          \tl_gput_right:Nx \g_@@_aux_tl
3839            {
```

```
3840            \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_cells_seq
3841              { \seq_use:Nnnn \g_@@_multicolumn_cells_seq , , , }
3842            \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_sizes_seq
3843              { \seq_use:Nnnn \g_@@_multicolumn_sizes_seq , , , }
3844          }
3845        }
```

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

```
3846        \@@_create_diag_nodes:
```

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

```
3847        \pgfpicture
3848        \int_step_inline:nn \c@iRow
3849          {
3850            \pgfnodealias
3851              { \@@_env: - ##1 - last }
3852              { \@@_env: - ##1 - \int_use:N \c@jCol }
3853          }
3854        \int_step_inline:nn \c@jCol
3855          {
3856            \pgfnodealias
3857              { \@@_env: - last - ##1 }
3858              { \@@_env: - \int_use:N \c@iRow - ##1 }
3859          }
3860        \str_if_empty:NF \l_@@_name_str
3861          {
3862            \int_step_inline:nn \c@iRow
3863              {
3864                \pgfnodealias
3865                  { \l_@@_name_str - ##1 - last }
3866                  { \@@_env: - ##1 - \int_use:N \c@jCol }
3867              }
3868            \int_step_inline:nn \c@jCol
3869              {
3870                \pgfnodealias
3871                  { \l_@@_name_str - last - ##1 }
3872                  { \@@_env: - \int_use:N \c@iRow - ##1 }
3873              }
3874          }
3875        \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.

```
3876        \bool_if:NT \l_@@_parallelize_diags_bool
3877          {
3878            \int_gzero_new:N \g_@@_ddots_int
3879            \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.

```
3880            \dim_gzero_new:N \g_@@_delta_x_one_dim
3881            \dim_gzero_new:N \g_@@_delta_y_one_dim
3882            \dim_gzero_new:N \g_@@_delta_x_two_dim
3883            \dim_gzero_new:N \g_@@_delta_y_two_dim
3884          }
3885        \int_zero_new:N \l_@@_initial_i_int
3886        \int_zero_new:N \l_@@_initial_j_int
```

---

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

```
3887        \int_zero_new:N \l_@@_final_i_int
3888        \int_zero_new:N \l_@@_final_j_int
3889        \bool_set_false:N \l_@@_initial_open_bool
3890        \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.

```
3891        \bool_if:NT \l_@@_small_bool
3892          {
3893            \dim_set:Nn \l_@@_xdots_radius_dim { 0.7 \l_@@_xdots_radius_dim }
3894            \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.

```
3895            \dim_set:Nn \l_@@_xdots_shorten_start_dim
3896              { 0.6 \l_@@_xdots_shorten_start_dim }
3897            \dim_set:Nn \l_@@_xdots_shorten_end_dim
3898              { 0.6 \l_@@_xdots_shorten_end_dim }
3899          }
```

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

```
3900        \@@_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.

```
3901        \@@_compute_corners:
```

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

```
3902        \@@_adjust_pos_of_blocks_seq:

3903        \@@_deal_with_rounded_corners:
3904        \tl_if_empty:NF \l_@@_hlines_clist \@@_draw_hlines:
3905        \tl_if_empty:NF \l_@@_vlines_clist \@@_draw_vlines:
```

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

```
3906        \IfPackageLoadedTF { tikz }
3907          {
3908            \tikzset
3909              {
3910                every~picture / .style =
3911                  {
3912                    overlay ,
3913                    remember~picture ,
3914                    name~prefix = \@@_env: -
3915                  }
3916              }
3917          }
3918          { }
3919        \bool_if:NT \c_@@_tagging_array_bool
3920          { \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign: }
3921        \cs_set_eq:NN \SubMatrix \@@_SubMatrix
3922        \cs_set_eq:NN \UnderBrace \@@_UnderBrace
3923        \cs_set_eq:NN \OverBrace \@@_OverBrace
3924        \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
3925        \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
3926        \cs_set_eq:NN \line \@@_line
3927        \g_@@_pre_code_after_tl
3928        \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.

```
3929        \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).

```
3930        \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).

```
3931        % \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
3932        %   { \@@_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:`.

```
3933        \bool_set_true:N \l_@@_in_code_after_bool
3934        \exp_last_unbraced:No \@@_CodeAfter_keys: \g_nicematrix_code_after_tl
3935        \scan_stop:
3936        \tl_gclear:N \g_nicematrix_code_after_tl
3937        \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.

```
3938        \seq_if_empty:NF \g_@@_rowlistcolors_seq { \@@_clear_rowlistcolors_seq: }
3939        \tl_if_empty:NF \g_@@_pre_code_before_tl
3940          {
3941            \tl_gput_right:Nx \g_@@_aux_tl
3942              {
3943                \tl_gset:Nn \exp_not:N \g_@@_pre_code_before_tl
3944                  { \exp_not:o \g_@@_pre_code_before_tl }
3945              }
3946            \tl_gclear:N \g_@@_pre_code_before_tl
3947          }
3948        \tl_if_empty:NF \g_nicematrix_code_before_tl
3949          {
3950            \tl_gput_right:Nx \g_@@_aux_tl
3951              {
3952                \tl_gset:Nn \exp_not:N \g_@@_code_before_tl
3953                  { \exp_not:o \g_nicematrix_code_before_tl }
3954              }
3955            \tl_gclear:N \g_nicematrix_code_before_tl
3956          }

3957        \str_gclear:N \g_@@_name_env_str
3958        \@@_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.

```
3959        \cs_gset_eq:NN \CT@arc@ \@@_old_CT@arc@
3960    }
```

---

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

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.

```
3961 \NewDocumentCommand \@@_CodeAfter_keys: { O { } }
3962   { \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).

```
3963 \cs_new_protected:Npn \@@_adjust_pos_of_blocks_seq:
3964   {
3965     \seq_gset_map_x:NNn \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq
3966       { \@@_adjust_pos_of_blocks_seq_i:nnnnn ##1 }
3967   }
```

The following command must *not* be protected.

```
3968 \cs_new:Npn \@@_adjust_pos_of_blocks_seq_i:nnnnn #1 #2 #3 #4 #5
3969   {
3970     { #1 }
3971     { #2 }
3972     {
3973       \int_compare:nNnTF { #3 } > { 99 }
3974         { \int_use:N \c@iRow }
3975         { #3 }
3976     }
3977     {
3978       \int_compare:nNnTF { #4 } > { 99 }
3979         { \int_use:N \c@jCol }
3980         { #4 }
3981     }
3982     { #5 }
3983   }
```

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).

```
3984 \hook_gput_code:nnn { begindocument } { . }
3985   {
3986     \cs_new_protected:Npx \@@_draw_dotted_lines:
3987       {
3988         \c_@@_pgfortikzpicture_tl
3989         \@@_draw_dotted_lines_i:
3990         \c_@@_endpgfortikzpicture_tl
3991       }
3992   }
```

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

```
3993 \cs_new_protected:Npn \@@_draw_dotted_lines_i:
3994   {
3995     \pgfrememberpicturepositiononpagetrue
3996     \pgf@relevantforpicturesizefalse
3997     \g_@@_HVdotsfor_lines_tl
3998     \g_@@_Vdots_lines_tl
3999     \g_@@_Ddots_lines_tl
4000     \g_@@_Iddots_lines_tl
4001     \g_@@_Cdots_lines_tl
4002     \g_@@_Ldots_lines_tl
4003   }
```

```
4004 \cs_new_protected:Npn \@@_restore_iRow_jCol:
4005   {
4006     \cs_if_exist:NT \theiRow { \int_gset_eq:NN \c@iRow \l_@@_old_iRow_int }
4007     \cs_if_exist:NT \thejCol { \int_gset_eq:NN \c@jCol \l_@@_old_jCol_int }
4008   }
```

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

```
4009 \pgfdeclareshape { @@_diag_node }
4010   {
4011     \savedanchor { \five }
4012       {
4013         \dim_gset_eq:NN \pgf@x \l_tmpa_dim
4014         \dim_gset_eq:NN \pgf@y \l_tmpb_dim
4015       }
4016     \anchor { 5 } { \five }
4017     \anchor { center } { \pgfpointorigin }
4018     \anchor { 1 } { \five \pgf@x = 0.2 \pgf@x \pgf@y = 0.2 \pgf@y }
4019     \anchor { 2 } { \five \pgf@x = 0.4 \pgf@x \pgf@y = 0.4 \pgf@y }
4020     \anchor { 3 } { \five \pgf@x = 0.6 \pgf@x \pgf@y = 0.6 \pgf@y }
4021     \anchor { 4 } { \five \pgf@x = 0.8 \pgf@x \pgf@y = 0.8 \pgf@y }
4022     \anchor { 6 } { \five \pgf@x = 1.2 \pgf@x \pgf@y = 1.2 \pgf@y }
4023     \anchor { 7 } { \five \pgf@x = 1.4 \pgf@x \pgf@y = 1.4 \pgf@y }
4024     \anchor { 8 } { \five \pgf@x = 1.6 \pgf@x \pgf@y = 1.6 \pgf@y }
4025     \anchor { 9 } { \five \pgf@x = 1.8 \pgf@x \pgf@y = 1.8 \pgf@y }
4026   }
```

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).

```
4027 \cs_new_protected:Npn \@@_create_diag_nodes:
4028   {
4029     \pgfpicture
4030     \pgfrememberpicturepositiononpagetrue
4031     \int_step_inline:nn { \int_max:nn \c@iRow \c@jCol }
4032       {
4033         \@@_qpoint:n { col - \int_min:nn { ##1 } { \c@jCol + 1 } }
4034         \dim_set_eq:NN \l_tmpa_dim \pgf@x
4035         \@@_qpoint:n { row - \int_min:nn { ##1 } { \c@iRow + 1 } }
4036         \dim_set_eq:NN \l_tmpb_dim \pgf@y
4037         \@@_qpoint:n { col - \int_min:nn { ##1 + 1 } { \c@jCol + 1 } }
4038         \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
4039         \@@_qpoint:n { row - \int_min:nn { ##1 + 1 } { \c@iRow + 1 } }
4040         \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
4041         \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.

```
4042         \dim_set:Nn \l_tmpa_dim { ( \l_@@_tmpc_dim - \l_tmpa_dim ) / 2 }
4043         \dim_set:Nn \l_tmpb_dim { ( \l_@@_tmpd_dim - \l_tmpb_dim ) / 2 }
4044         \pgfnode { @@_diag_node } { center } { } { \@@_env: - ##1 } { }
4045         \str_if_empty:NF \l_@@_name_str
4046           { \pgfnodealias { \l_@@_name_str - ##1 } { \@@_env: - ##1 } }
4047       }
```

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

```
4048     \int_set:Nn \l_tmpa_int { \int_max:nn \c@iRow \c@jCol + 1 }
4049     \@@_qpoint:n { row - \int_min:nn { \l_tmpa_int } { \c@iRow + 1 } }
4050     \dim_set_eq:NN \l_tmpa_dim \pgf@y
4051     \@@_qpoint:n { col - \int_min:nn { \l_tmpa_int } { \c@jCol + 1 } }
4052     \pgfcoordinate
4053       { \@@_env: - \int_use:N \l_tmpa_int } { \pgfpoint \pgf@x \l_tmpa_dim }
4054     \pgfnodealias
```

```
4055        { \@@_env: - last }
4056        { \@@_env: - \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
4057      \str_if_empty:NF \l_@@_name_str
4058        {
4059          \pgfnodealias
4060            { \l_@@_name_str - \int_use:N \l_tmpa_int }
4061            { \@@_env: - \int_use:N \l_tmpa_int }
4062          \pgfnodealias
4063            { \l_@@_name_str - last }
4064            { \@@_env: - last }
4065        }
4066      \endpgfpicture
4067    }
```

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.

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

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.

```
4068 \cs_new_protected:Npn \@@_find_extremities_of_line:nnnn #1 #2 #3 #4
4069    {
```

First, we declare the current cell as "dotted" because we forbide intersections of dotted lines.

```
4070      \cs_set:cpn { @@ _ dotted _ #1 - #2 } { }
```

Initialization of variables.

```
4071      \int_set:Nn \l_@@_initial_i_int { #1 }
4072      \int_set:Nn \l_@@_initial_j_int { #2 }
4073      \int_set:Nn \l_@@_final_i_int { #1 }
4074      \int_set:Nn \l_@@_final_j_int { #2 }
```

100

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.

```
4075        \bool_set_false:N \l_@@_stop_loop_bool
4076        \bool_do_until:Nn \l_@@_stop_loop_bool
4077          {
4078            \int_add:Nn \l_@@_final_i_int { #3 }
4079            \int_add:Nn \l_@@_final_j_int { #4 }
```

We test if we are still in the matrix.

```
4080            \bool_set_false:N \l_@@_final_open_bool
4081            \int_compare:nNnTF \l_@@_final_i_int > \l_@@_row_max_int
4082              {
4083                \int_compare:nNnTF { #3 } = \c_one_int
4084                  { \bool_set_true:N \l_@@_final_open_bool }
4085                  {
4086                    \int_compare:nNnT \l_@@_final_j_int > \l_@@_col_max_int
4087                      { \bool_set_true:N \l_@@_final_open_bool }
4088                  }
4089              }
4090              {
4091                \int_compare:nNnTF \l_@@_final_j_int < \l_@@_col_min_int
4092                  {
4093                    \int_compare:nNnT { #4 } = { -1 }
4094                      { \bool_set_true:N \l_@@_final_open_bool }
4095                  }
4096                  {
4097                    \int_compare:nNnT \l_@@_final_j_int > \l_@@_col_max_int
4098                      {
4099                        \int_compare:nNnT { #4 } = \c_one_int
4100                          { \bool_set_true:N \l_@@_final_open_bool }
4101                      }
4102                  }
4103              }
4104            \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.

```
4105              {
```

We do a step backwards.

```
4106                \int_sub:Nn \l_@@_final_i_int { #3 }
4107                \int_sub:Nn \l_@@_final_j_int { #4 }
4108                \bool_set_true:N \l_@@_stop_loop_bool
4109              }
```

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`.

```
4110              {
4111                \cs_if_exist:cTF
4112                  {
4113                    @@ _ dotted _
4114                    \int_use:N \l_@@_final_i_int -
4115                    \int_use:N \l_@@_final_j_int
4116                  }
4117                  {
4118                    \int_sub:Nn \l_@@_final_i_int { #3 }
4119                    \int_sub:Nn \l_@@_final_j_int { #4 }
4120                    \bool_set_true:N \l_@@_final_open_bool
4121                    \bool_set_true:N \l_@@_stop_loop_bool
4122                  }
4123                  {
4124                    \cs_if_exist:cTF
4125                      {
4126                        pgf @ sh @ ns @ \@@_env:
4127                        - \int_use:N \l_@@_final_i_int
```

```
4128                        - \int_use:N \l_@@_final_j_int
4129                      }
4130                    { \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.

```
4131                    {
4132                      \cs_set:cpn
4133                        {
4134                          @@ _ dotted _
4135                          \int_use:N \l_@@_final_i_int -
4136                          \int_use:N \l_@@_final_j_int
4137                        }
4138                        { }
4139                    }
4140                }
4141            }
4142        }
```

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

```
4143        \bool_set_false:N \l_@@_stop_loop_bool
4144        \bool_do_until:Nn \l_@@_stop_loop_bool
4145          {
4146            \int_sub:Nn \l_@@_initial_i_int { #3 }
4147            \int_sub:Nn \l_@@_initial_j_int { #4 }
4148            \bool_set_false:N \l_@@_initial_open_bool
4149            \int_compare:nNnTF \l_@@_initial_i_int < \l_@@_row_min_int
4150              {
4151                \int_compare:nNnTF { #3 } = \c_one_int
4152                  { \bool_set_true:N \l_@@_initial_open_bool }
4153                  {
4154                    \int_compare:nNnT \l_@@_initial_j_int = { \l_@@_col_min_int - 1 }
4155                      { \bool_set_true:N \l_@@_initial_open_bool }
4156                  }
4157              }
4158              {
4159                \int_compare:nNnTF \l_@@_initial_j_int < \l_@@_col_min_int
4160                  {
4161                    \int_compare:nNnT { #4 } = \c_one_int
4162                      { \bool_set_true:N \l_@@_initial_open_bool }
4163                  }
4164                  {
4165                    \int_compare:nNnT \l_@@_initial_j_int > \l_@@_col_max_int
4166                      {
4167                        \int_compare:nNnT { #4 } = { -1 }
4168                          { \bool_set_true:N \l_@@_initial_open_bool }
4169                      }
4170                  }
4171              }
4172            \bool_if:NTF \l_@@_initial_open_bool
4173              {
4174                \int_add:Nn \l_@@_initial_i_int { #3 }
4175                \int_add:Nn \l_@@_initial_j_int { #4 }
4176                \bool_set_true:N \l_@@_stop_loop_bool
4177              }
4178              {
4179                \cs_if_exist:cTF
4180                  {
```

```
4181              @@ _ dotted _
4182              \int_use:N \l_@@_initial_i_int -
4183              \int_use:N \l_@@_initial_j_int
4184            }
4185            {
4186              \int_add:Nn \l_@@_initial_i_int { #3 }
4187              \int_add:Nn \l_@@_initial_j_int { #4 }
4188              \bool_set_true:N \l_@@_initial_open_bool
4189              \bool_set_true:N \l_@@_stop_loop_bool
4190            }
4191            {
4192              \cs_if_exist:cTF
4193                {
4194                  pgf @ sh @ ns @ \@@_env:
4195                  - \int_use:N \l_@@_initial_i_int
4196                  - \int_use:N \l_@@_initial_j_int
4197                }
4198                { \bool_set_true:N \l_@@_stop_loop_bool }
4199                {
4200                  \cs_set:cpn
4201                    {
4202                      @@ _ dotted _
4203                      \int_use:N \l_@@_initial_i_int -
4204                      \int_use:N \l_@@_initial_j_int
4205                    }
4206                    { }
4207                }
4208            }
4209          }
4210        }
```

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.

```
4211        \seq_gput_right:Nx \g_@@_pos_of_xdots_seq
4212          {
4213            { \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`.

```
4214            { \int_min:nn \l_@@_initial_j_int \l_@@_final_j_int }
4215            { \int_use:N \l_@@_final_i_int }
4216            { \int_max:nn \l_@@_initial_j_int \l_@@_final_j_int }
4217            { } % for the name of the block
4218          }
4219    }
```

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.

```
4220 \cs_new_protected:Npn \@@_open_shorten:
4221   {
4222     \bool_if:NT \l_@@_initial_open_bool
4223       { \dim_zero:N \l_@@_xdots_shorten_start_dim }
4224     \bool_if:NT \l_@@_final_open_bool
4225       { \dim_zero:N \l_@@_xdots_shorten_end_dim }
4226   }
```

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).

```
4227  \cs_new_protected:Npn \@@_adjust_to_submatrix:nn #1 #2
4228    {
4229      \int_set:Nn \l_@@_row_min_int 1
4230      \int_set:Nn \l_@@_col_min_int 1
4231      \int_set_eq:NN \l_@@_row_max_int \c@iRow
4232      \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`.

```
4233      \seq_map_inline:Nn \g_@@_submatrix_seq
4234        { \@@_adjust_to_submatrix:nnnnnn { #1 } { #2 } ##1 }
4235    }
```

`#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.

```
4236  \cs_set_protected:Npn \@@_adjust_to_submatrix:nnnnnn #1 #2 #3 #4 #5 #6
4237    {
4238      \int_compare:nNnF { #3 } > { #1 }
4239        {
4240          \int_compare:nNnF { #1 } > {  #5 }
4241            {
4242              \int_compare:nNnF { #4 } > { #2 }
4243                {
4244                  \int_compare:nNnF { #2 } > { #6 }
4245                    {
4246                      \int_set:Nn \l_@@_row_min_int
4247                        { \int_max:nn \l_@@_row_min_int { #3 } }
4248                      \int_set:Nn \l_@@_col_min_int
4249                        { \int_max:nn \l_@@_col_min_int { #4 } }
4250                      \int_set:Nn \l_@@_row_max_int
4251                        { \int_min:nn \l_@@_row_max_int { #5 } }
4252                      \int_set:Nn \l_@@_col_max_int
4253                        { \int_min:nn \l_@@_col_max_int { #6 } }
4254                    }
4255                }
4256            }
4257        }
4258    }


4259  \cs_new_protected:Npn \@@_set_initial_coords:
4260    {
4261      \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4262      \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4263    }
4264  \cs_new_protected:Npn \@@_set_final_coords:
4265    {
4266      \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4267      \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4268    }
4269  \cs_new_protected:Npn \@@_set_initial_coords_from_anchor:n #1
4270    {
4271      \pgfpointanchor
4272        {
4273          \@@_env:
4274          - \int_use:N \l_@@_initial_i_int
4275          - \int_use:N \l_@@_initial_j_int
4276        }
4277        { #1 }
4278      \@@_set_initial_coords:
4279    }
4280  \cs_new_protected:Npn \@@_set_final_coords_from_anchor:n #1
4281    {
```

```
4282        \pgfpointanchor
4283          {
4284            \@@_env:
4285            - \int_use:N \l_@@_final_i_int
4286            - \int_use:N \l_@@_final_j_int
4287          }
4288        { #1 }
4289      \@@_set_final_coords:
4290    }
4291  \cs_new_protected:Npn \@@_open_x_initial_dim:
4292    {
4293      \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
4294      \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
4295        {
4296          \cs_if_exist:cT
4297            { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4298            {
4299              \pgfpointanchor
4300                { \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
4301                { west }
4302              \dim_set:Nn \l_@@_x_initial_dim
4303                { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
4304            }
4305        }
```

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

```
4306      \dim_compare:nNnT \l_@@_x_initial_dim = \c_max_dim
4307        {
4308          \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4309          \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4310          \dim_add:Nn \l_@@_x_initial_dim \col@sep
4311        }
4312    }
4313  \cs_new_protected:Npn \@@_open_x_final_dim:
4314    {
4315      \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
4316      \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
4317        {
4318          \cs_if_exist:cT
4319            { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4320            {
4321              \pgfpointanchor
4322                { \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
4323                { east }
4324              \dim_set:Nn \l_@@_x_final_dim
4325                { \dim_max:nn \l_@@_x_final_dim \pgf@x }
4326            }
4327        }
```

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

```
4328      \dim_compare:nNnT \l_@@_x_final_dim = { - \c_max_dim }
4329        {
4330          \@@_qpoint:n { col - \int_eval:n { \l_@@_final_j_int + 1 } }
4331          \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4332          \dim_sub:Nn \l_@@_x_final_dim \col@sep
4333        }
4334    }
```

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.

```
4335  \cs_new_protected:Npn \@@_draw_Ldots:nnn #1 #2 #3
4336    {
```

```
4337        \@@_adjust_to_submatrix:nn { #1 } { #2 }
4338        \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4339          {
4340            \@@_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.

```
4341            \group_begin:
4342              \@@_open_shorten:
4343              \int_if_zero:nTF { #1 }
4344                { \color { nicematrix-first-row } }
4345                {
```

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.

```
4346                  \int_compare:nNnT { #1 } = \l_@@_last_row_int
4347                    { \color { nicematrix-last-row } }
4348                }
4349              \keys_set:nn { NiceMatrix / xdots } { #3 }
4350              \tl_if_empty:oF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4351              \@@_actually_draw_Ldots:
4352            \group_end:
4353          }
4354    }
```

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.

```
4355 \cs_new_protected:Npn \@@_actually_draw_Ldots:
4356    {
4357      \bool_if:NTF \l_@@_initial_open_bool
4358        {
4359          \@@_open_x_initial_dim:
4360          \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4361          \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4362        }
4363        { \@@_set_initial_coords_from_anchor:n { base~east } }
4364      \bool_if:NTF \l_@@_final_open_bool
4365        {
4366          \@@_open_x_final_dim:
4367          \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4368          \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4369        }
4370        { \@@_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.

```
4371        \bool_lazy_all:nTF
4372          {
4373            \l_@@_initial_open_bool
4374            \l_@@_final_open_bool
4375            { \int_compare_p:nNn \l_@@_initial_i_int = \l_@@_last_row_int }
4376          }
```

```
4377          {
4378            \dim_add:Nn \l_@@_y_initial_dim \c_@@_shift_Ldots_last_row_dim
4379            \dim_add:Nn \l_@@_y_final_dim \c_@@_shift_Ldots_last_row_dim
4380          }
```

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 (?).

```
4381          {
4382            \dim_add:Nn \l_@@_y_initial_dim \l_@@_xdots_radius_dim
4383            \dim_add:Nn \l_@@_y_final_dim \l_@@_xdots_radius_dim
4384          }
4385        \@@_draw_line:
4386      }
```

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.

```
4387  \cs_new_protected:Npn \@@_draw_Cdots:nnn #1 #2 #3
4388    {
4389      \@@_adjust_to_submatrix:nn { #1 } { #2 }
4390      \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4391        {
4392          \@@_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.

```
4393          \group_begin:
4394            \@@_open_shorten:
4395            \int_if_zero:nTF { #1 }
4396              { \color { nicematrix-first-row } }
4397              {
```

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.

```
4398                \int_compare:nNnT { #1 } = \l_@@_last_row_int
4399                  { \color { nicematrix-last-row } }
4400              }
4401            \keys_set:nn { NiceMatrix / xdots } { #3 }
4402            \tl_if_empty:oF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4403            \@@_actually_draw_Cdots:
4404          \group_end:
4405        }
4406    }
```

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.

```
4407  \cs_new_protected:Npn \@@_actually_draw_Cdots:
4408    {
4409      \bool_if:NTF \l_@@_initial_open_bool
4410        { \@@_open_x_initial_dim: }
4411        { \@@_set_initial_coords_from_anchor:n { mid~east } }
4412      \bool_if:NTF \l_@@_final_open_bool
4413        { \@@_open_x_final_dim: }
4414        { \@@_set_final_coords_from_anchor:n { mid~west } }
```

```
4415      \bool_lazy_and:nnTF
4416        \l_@@_initial_open_bool
4417        \l_@@_final_open_bool
4418        {
4419          \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int }
4420          \dim_set_eq:NN \l_tmpa_dim \pgf@y
4421          \@@_qpoint:n { row - \int_eval:n { \l_@@_initial_i_int + 1 } }
4422          \dim_set:Nn \l_@@_y_initial_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
4423          \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
4424        }
4425        {
4426          \bool_if:NT \l_@@_initial_open_bool
4427            { \dim_set_eq:NN \l_@@_y_initial_dim \l_@@_y_final_dim }
4428          \bool_if:NT \l_@@_final_open_bool
4429            { \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim }
4430        }
4431      \@@_draw_line:
4432    }
4433  \cs_new_protected:Npn \@@_open_y_initial_dim:
4434    {
4435      \dim_set:Nn \l_@@_y_initial_dim { - \c_max_dim }
4436      \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
4437        {
4438          \cs_if_exist:cT
4439            { pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4440            {
4441              \pgfpointanchor
4442                { \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
4443                { north }
4444              \dim_set:Nn \l_@@_y_initial_dim
4445                { \dim_max:nn \l_@@_y_initial_dim \pgf@y }
4446            }
4447        }
4448      \dim_compare:nNnT \l_@@_y_initial_dim = { - \c_max_dim }
4449        {
4450          \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
4451          \dim_set:Nn \l_@@_y_initial_dim
4452            {
4453              \fp_to_dim:n
4454                {
4455                  \pgf@y
4456                  + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
4457                }
4458            }
4459        }
4460    }
4461  \cs_new_protected:Npn \@@_open_y_final_dim:
4462    {
4463      \dim_set_eq:NN \l_@@_y_final_dim \c_max_dim
4464      \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
4465        {
4466          \cs_if_exist:cT
4467            { pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4468            {
4469              \pgfpointanchor
4470                { \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
4471                { south }
4472              \dim_set:Nn \l_@@_y_final_dim
4473                { \dim_min:nn \l_@@_y_final_dim \pgf@y }
4474            }
4475        }
4476      \dim_compare:nNnT \l_@@_y_final_dim = \c_max_dim
4477        {
```

```
4478          \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
4479          \dim_set:Nn \l_@@_y_final_dim
4480            { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } } }
4481        }
4482    }
```

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.

```
4483  \cs_new_protected:Npn \@@_draw_Vdots:nnn #1 #2 #3
4484    {
4485      \@@_adjust_to_submatrix:nn { #1 } { #2 }
4486      \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4487        {
4488          \@@_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.

```
4489          \group_begin:
4490            \@@_open_shorten:
4491            \int_if_zero:nTF { #2 }
4492              { \color { nicematrix-first-col } }
4493              {
4494                \int_compare:nNnT { #2 } = \l_@@_last_col_int
4495                  { \color { nicematrix-last-col } }
4496              }
4497            \keys_set:nn { NiceMatrix / xdots } { #3 }
4498            \tl_if_empty:oF \l_@@_xdots_color_tl
4499              { \color { \l_@@_xdots_color_tl } }
4500            \@@_actually_draw_Vdots:
4501          \group_end:
4502        }
4503    }
```

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`.

```
4504  \cs_new_protected:Npn \@@_actually_draw_Vdots:
4505    {
```

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

```
4506      \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.

```
4507        {
4508          \@@_open_y_initial_dim:
4509          \@@_open_y_final_dim:
4510          \int_if_zero:nTF \l_@@_initial_j_int
```

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

```
4511            {
4512              \@@_qpoint:n { col - 1 }
4513              \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4514              \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
4515              \dim_sub:Nn \l_@@_x_initial_dim \l_@@_extra_left_margin_dim
```

```
4516                \dim_sub:Nn \l_@@_x_initial_dim \c_@@_shift_exterior_Vdots_dim
4517              }
4518              {
4519                \bool_lazy_and:nnTF
4520                  { \int_compare_p:nNn \l_@@_last_col_int > { -2 } }
4521                  { \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".

```
4522                  {
4523                    \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4524                    \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4525                    \dim_add:Nn \l_@@_x_initial_dim \l_@@_right_margin_dim
4526                    \dim_add:Nn \l_@@_x_initial_dim \l_@@_extra_right_margin_dim
4527                    \dim_add:Nn \l_@@_x_initial_dim \c_@@_shift_exterior_Vdots_dim
4528                  }
```

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

```
4529                  {
4530                    \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
4531                    \dim_set_eq:NN \l_tmpa_dim \pgf@x
4532                    \@@_qpoint:n { col - \int_eval:n { \l_@@_initial_j_int + 1 } }
4533                    \dim_set:Nn \l_@@_x_initial_dim { ( \pgf@x + \l_tmpa_dim ) / 2 }
4534                  }
4535                }
4536            }
```

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.

```
4537          {
4538            \bool_set_false:N \l_tmpa_bool
4539            \bool_if:NF \l_@@_initial_open_bool
4540              {
4541                \bool_if:NF \l_@@_final_open_bool
4542                  {
4543                    \@@_set_initial_coords_from_anchor:n { south~west }
4544                    \@@_set_final_coords_from_anchor:n { north~west }
4545                    \bool_set:Nn \l_tmpa_bool
4546                      { \dim_compare_p:nNn \l_@@_x_initial_dim = \l_@@_x_final_dim }
4547                  }
4548              }
```

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

```
4549            \bool_if:NTF \l_@@_initial_open_bool
4550              {
4551                \@@_open_y_initial_dim:
4552                \@@_set_final_coords_from_anchor:n { north }
4553                \dim_set_eq:NN \l_@@_x_initial_dim \l_@@_x_final_dim
4554              }
4555              {
4556                \@@_set_initial_coords_from_anchor:n { south }
4557                \bool_if:NTF \l_@@_final_open_bool
4558                  \@@_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.

```
4559                  {
4560                    \@@_set_final_coords_from_anchor:n { north }
4561                    \dim_compare:nNnF \l_@@_x_initial_dim = \l_@@_x_final_dim
4562                      {
4563                        \dim_set:Nn \l_@@_x_initial_dim
4564                          {
4565                            \bool_if:NTF \l_tmpa_bool \dim_min:nn \dim_max:nn
4566                              \l_@@_x_initial_dim \l_@@_x_final_dim
4567                          }
4568                      }
```

```
4569                    }
4570                }
4571            }
4572        \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
4573        \@@_draw_line:
4574    }
```

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.

```
4575 \cs_new_protected:Npn \@@_draw_Ddots:nnn #1 #2 #3
4576    {
4577        \@@_adjust_to_submatrix:nn { #1 } { #2 }
4578        \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4579            {
4580                \@@_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.

```
4581                \group_begin:
4582                    \@@_open_shorten:
4583                    \keys_set:nn { NiceMatrix / xdots } { #3 }
4584                    \tl_if_empty:oF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4585                    \@@_actually_draw_Ddots:
4586                \group_end:
4587            }
4588    }
```

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`.

```
4589 \cs_new_protected:Npn \@@_actually_draw_Ddots:
4590    {
4591        \bool_if:NTF \l_@@_initial_open_bool
4592            {
4593                \@@_open_y_initial_dim:
4594                \@@_open_x_initial_dim:
4595            }
4596            { \@@_set_initial_coords_from_anchor:n { south~east } }
4597        \bool_if:NTF \l_@@_final_open_bool
4598            {
4599                \@@_open_x_final_dim:
4600                \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4601            }
4602            { \@@_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.

```
4603        \bool_if:NT \l_@@_parallelize_diags_bool
4604            {
4605                \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).

```
4606          \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.

```
4607            {
4608              \dim_gset:Nn \g_@@_delta_x_one_dim
4609                { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4610              \dim_gset:Nn \g_@@_delta_y_one_dim
4611                { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4612            }
```

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`.

```
4613            {
4614              \dim_set:Nn \l_@@_y_final_dim
4615                {
4616                  \l_@@_y_initial_dim +
4617                  ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4618                  \dim_ratio:nn \g_@@_delta_y_one_dim \g_@@_delta_x_one_dim
4619                }
4620            }
4621          }
4622      \@@_draw_line:
4623    }
```

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.

```
4624 \cs_new_protected:Npn \@@_draw_Iddots:nnn #1 #2 #3
4625    {
4626      \@@_adjust_to_submatrix:nn { #1 } { #2 }
4627      \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
4628        {
4629          \@@_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.

```
4630          \group_begin:
4631            \@@_open_shorten:
4632            \keys_set:nn { NiceMatrix / xdots } { #3 }
4633            \tl_if_empty:oF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4634            \@@_actually_draw_Iddots:
4635          \group_end:
4636        }
4637    }
```

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`.

```
4638  \cs_new_protected:Npn \@@_actually_draw_Iddots:
4639    {
4640      \bool_if:NTF \l_@@_initial_open_bool
4641        {
4642          \@@_open_y_initial_dim:
4643          \@@_open_x_initial_dim:
4644        }
4645        { \@@_set_initial_coords_from_anchor:n { south~west } }
4646      \bool_if:NTF \l_@@_final_open_bool
4647        {
4648          \@@_open_y_final_dim:
4649          \@@_open_x_final_dim:
4650        }
4651        { \@@_set_final_coords_from_anchor:n { north~east } }
4652      \bool_if:NT \l_@@_parallelize_diags_bool
4653        {
4654          \int_gincr:N \g_@@_iddots_int
4655          \int_compare:nNnTF \g_@@_iddots_int = \c_one_int
4656            {
4657              \dim_gset:Nn \g_@@_delta_x_two_dim
4658                { \l_@@_x_final_dim - \l_@@_x_initial_dim }
4659              \dim_gset:Nn \g_@@_delta_y_two_dim
4660                { \l_@@_y_final_dim - \l_@@_y_initial_dim }
4661            }
4662            {
4663              \dim_set:Nn \l_@@_y_final_dim
4664                {
4665                  \l_@@_y_initial_dim +
4666                  ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4667                  \dim_ratio:nn \g_@@_delta_y_two_dim \g_@@_delta_x_two_dim
4668                }
4669            }
4670        }
4671      \@@_draw_line:
4672    }
```

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`

```
4673  \cs_new_protected:Npn \@@_draw_line:
4674    {
4675      \pgfrememberpicturepositiononpagetrue
4676      \pgf@relevantforpicturesizefalse
4677      \bool_lazy_or:nnTF
4678        { \tl_if_eq_p:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl }
4679        \l_@@_dotted_bool
```

```
4680        \@@_draw_standard_dotted_line:
4681        \@@_draw_unstandard_dotted_line:
4682    }
```

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.

```
4683  \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:
4684    {
4685      \begin { scope }
4686      \@@_draw_unstandard_dotted_line:o
4687        { \l_@@_xdots_line_style_tl , \l_@@_xdots_color_tl }
4688    }
```

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.

```
4689  \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:n #1
4690    {
4691      \@@_draw_unstandard_dotted_line:nooo
4692        { #1 }
4693      \l_@@_xdots_up_tl
4694      \l_@@_xdots_down_tl
4695      \l_@@_xdots_middle_tl
4696    }
4697  \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.

```
4698  \hook_gput_code:nnn { begindocument } { . }
4699    {
4700      \IfPackageLoadedTF { tikz }
4701        {
4702          \tikzset
4703            {
4704              @@_node_above / .style = { sloped , above } ,
4705              @@_node_below / .style = { sloped , below } ,
4706              @@_node_middle / .style =
4707                {
4708                  sloped ,
4709                  inner~sep = \c_@@_innersep_middle_dim
4710                }
4711            }
4712        }
4713        { }
4714    }
```

```
4715  \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:nnnn #1 #2 #3 #4
4716    {
```

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.

```
4717        \dim_zero_new:N \l_@@_l_dim
4718        \dim_set:Nn \l_@@_l_dim
4719          {
4720            \fp_to_dim:n
4721              {
4722                sqrt
4723                  (
```

```
4724            ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
4725                  +
4726            ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
4727          )
4728        }
4729      }
```

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.

```
4730      \dim_compare:nNnT \l_@@_l_dim < \c_@@_max_l_dim
4731        {
4732          \dim_compare:nNnT \l_@@_l_dim  > { 1 pt }
4733            \@@_draw_unstandard_dotted_line_i:
4734        }
```

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

```
4735      \bool_if:NT \l_@@_xdots_h_labels_bool
4736        {
4737          \tikzset
4738            {
4739              @@_node_above / .style = { auto = left } ,
4740              @@_node_below / .style = { auto = right } ,
4741              @@_node_middle / .style = { inner~sep = \c_@@_innersep_middle_dim }
4742            }
4743        }
4744      \tl_if_empty:nF { #4 }
4745        { \tikzset { @@_node_middle / .append~style = { fill = white } } }
4746      \draw
4747        [ #1 ]
4748          ( \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).

```
4749        -- node [ @@_node_middle] { $ \scriptstyle #4 $ }
4750           node [ @@_node_below ] { $ \scriptstyle #3 $ }
4751           node [ @@_node_above ] { $ \scriptstyle #2 $ }
4752           ( \l_@@_x_final_dim , \l_@@_y_final_dim ) ;
4753      \end { scope }
4754    }
4755  \cs_new_protected:Npn \@@_draw_unstandard_dotted_line_i:
4756    {
4757      \dim_set:Nn \l_tmpa_dim
4758        {
4759          \l_@@_x_initial_dim
4760          + ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
4761          * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
4762        }
4763      \dim_set:Nn \l_tmpb_dim
4764        {
4765          \l_@@_y_initial_dim
4766          + ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
4767          * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
4768        }
4769      \dim_set:Nn \l_@@_tmpc_dim
4770        {
4771          \l_@@_x_final_dim
4772          - ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
4773          * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
4774        }
4775      \dim_set:Nn \l_@@_tmpd_dim
4776        {
4777          \l_@@_y_final_dim
```

```
4778              - ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
4779              * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
4780            }
4781        \dim_set_eq:NN \l_@@_x_initial_dim \l_tmpa_dim
4782        \dim_set_eq:NN \l_@@_y_initial_dim \l_tmpb_dim
4783        \dim_set_eq:NN \l_@@_x_final_dim \l_@@_tmpc_dim
4784        \dim_set_eq:NN \l_@@_y_final_dim \l_@@_tmpd_dim
4785    }
4786 \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).

```
4787 \cs_new_protected:Npn \@@_draw_standard_dotted_line:
4788    {
4789        \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.

```
4790        \dim_zero_new:N \l_@@_l_dim
4791        \dim_set:Nn \l_@@_l_dim
4792          {
4793            \fp_to_dim:n
4794              {
4795                sqrt
4796                  (
4797                    ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
4798                      +
4799                    ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
4800                  )
4801              }
4802          }
```

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.

```
4803        \dim_compare:nNnT \l_@@_l_dim < \c_@@_max_l_dim
4804          {
4805            \dim_compare:nNnT \l_@@_l_dim  > { 1 pt }
4806              \@@_draw_standard_dotted_line_i:
4807          }
4808        \group_end:
4809        \bool_lazy_all:nF
4810          {
4811            { \tl_if_empty_p:N \l_@@_xdots_up_tl }
4812            { \tl_if_empty_p:N \l_@@_xdots_down_tl }
4813            { \tl_if_empty_p:N \l_@@_xdots_middle_tl }
4814          }
4815        \l_@@_labels_standard_dotted_line:
4816    }
4817 \dim_const:Nn \c_@@_max_l_dim { 50 cm }
4818 \cs_new_protected:Npn \@@_draw_standard_dotted_line_i:
4819    {
```

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

```
4820        \int_set:Nn \l_tmpa_int
4821          {
4822            \dim_ratio:nn
4823              {
4824                \l_@@_l_dim
4825                - \l_@@_xdots_shorten_start_dim
4826                - \l_@@_xdots_shorten_end_dim
```

```
4827              }
4828          \l_@@_xdots_inter_dim
4829      }
```

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

```
4830      \dim_set:Nn \l_tmpa_dim
4831        {
4832          ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4833          \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
4834        }
4835      \dim_set:Nn \l_tmpb_dim
4836        {
4837          ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
4838          \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
4839        }
```

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.

```
4840      \dim_gadd:Nn \l_@@_x_initial_dim
4841        {
4842          ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
4843          \dim_ratio:nn
4844            {
4845              \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
4846              + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
4847            }
4848            { 2 \l_@@_l_dim }
4849        }
4850      \dim_gadd:Nn \l_@@_y_initial_dim
4851        {
4852          ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
4853          \dim_ratio:nn
4854            {
4855              \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
4856              + \l_@@_xdots_shorten_start_dim  - \l_@@_xdots_shorten_end_dim
4857            }
4858            { 2 \l_@@_l_dim }
4859        }
4860      \pgf@relevantforpicturesizefalse
4861      \int_step_inline:nnn \c_zero_int \l_tmpa_int
4862        {
4863          \pgfpathcircle
4864            { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
4865            { \l_@@_xdots_radius_dim }
4866          \dim_add:Nn \l_@@_x_initial_dim \l_tmpa_dim
4867          \dim_add:Nn \l_@@_y_initial_dim \l_tmpb_dim
4868        }
4869      \pgfusepathqfill
4870    }


4871 \cs_new_protected:Npn \l_@@_labels_standard_dotted_line:
4872    {
4873      \pgfscope
4874      \pgftransformshift
4875        {
4876          \pgfpointlineattime { 0.5 }
4877            { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
4878            { \pgfpoint \l_@@_x_final_dim \l_@@_y_final_dim }
4879        }
4880      \fp_set:Nn \l_tmpa_fp
4881        {
4882          atand
```

```
                (
                  \l_@@_y_final_dim - \l_@@_y_initial_dim ,
                  \l_@@_x_final_dim - \l_@@_x_initial_dim
                )
            }
        \pgftransformrotate { \fp_use:N \l_tmpa_fp }
        \bool_if:NF \l_@@_xdots_h_labels_bool { \fp_zero:N \l_tmpa_fp }
        \tl_if_empty:NF \l_@@_xdots_middle_tl
          {
            \begin { pgfscope }
            \pgfset { inner~sep = \c_@@_innersep_middle_dim }
            \pgfnode
              { rectangle }
              { center }
              {
                \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
                  {
                    \c_math_toggle_token
                    \scriptstyle \l_@@_xdots_middle_tl
                    \c_math_toggle_token
                  }
              }
              { }
              {
                \pgfsetfillcolor { white }
                \pgfusepath { fill }
              }
            \end { pgfscope }
          }
        \tl_if_empty:NF \l_@@_xdots_up_tl
          {
            \pgfnode
              { rectangle }
              { south }
              {
                \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
                  {
                    \c_math_toggle_token
                    \scriptstyle \l_@@_xdots_up_tl
                    \c_math_toggle_token
                  }
              }
              { }
              { \pgfusepath { } }
          }
        \tl_if_empty:NF \l_@@_xdots_down_tl
          {
            \pgfnode
              { rectangle }
              { north }
              {
                \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
                  {
                    \c_math_toggle_token
                    \scriptstyle \l_@@_xdots_down_tl
                    \c_math_toggle_token
                  }
              }
              { }
              { \pgfusepath { } }
          }
      \endpgfscope
  }
```

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.

```
4946  \hook_gput_code:nnn { begindocument } { . }
4947    {
4948      \cs_set_nopar:Npn \l_@@_argspec_tl { m E { _ ^ : } { { } { } { } } }
4949      \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
4950      \cs_new_protected:Npn \@@_Ldots
4951        { \@@_collect_options:n { \@@_Ldots_i } }
4952      \exp_args:NNo \NewDocumentCommand \@@_Ldots_i \l_@@_argspec_tl
4953        {
4954          \int_if_zero:nTF \c@jCol
4955            { \@@_error:nn { in~first~col } \Ldots }
4956            {
4957              \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
4958                { \@@_error:nn { in~last~col } \Ldots }
4959                {
4960                  \@@_instruction_of_type:nnn \c_false_bool { Ldots }
4961                    { #1 , down = #2 , up = #3 , middle = #4 }
4962                }
4963            }
4964          \bool_if:NF \l_@@_nullify_dots_bool
4965            { \phantom { \ensuremath { \@@_old_ldots } } }
4966          \bool_gset_true:N \g_@@_empty_cell_bool
4967        }

4968      \cs_new_protected:Npn \@@_Cdots
4969        { \@@_collect_options:n { \@@_Cdots_i } }
4970      \exp_args:NNo \NewDocumentCommand \@@_Cdots_i \l_@@_argspec_tl
4971        {
4972          \int_if_zero:nTF \c@jCol
4973            { \@@_error:nn { in~first~col } \Cdots }
4974            {
4975              \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
4976                { \@@_error:nn { in~last~col } \Cdots }
4977                {
4978                  \@@_instruction_of_type:nnn \c_false_bool { Cdots }
4979                    { #1 , down = #2 , up = #3 , middle = #4 }
4980                }
4981            }
4982          \bool_if:NF \l_@@_nullify_dots_bool
4983            { \phantom { \ensuremath { \@@_old_cdots } } }
4984          \bool_gset_true:N \g_@@_empty_cell_bool
4985        }

4986      \cs_new_protected:Npn \@@_Vdots
4987        { \@@_collect_options:n { \@@_Vdots_i } }
4988      \exp_args:NNo \NewDocumentCommand \@@_Vdots_i \l_@@_argspec_tl
4989        {
4990          \int_if_zero:nTF \c@iRow
4991            { \@@_error:nn { in~first~row } \Vdots }
4992            {
```

```
4993        \int_compare:nNnTF \c@iRow = \l_@@_last_row_int
4994          { \@@_error:nn { in~last~row } \Vdots }
4995          {
4996            \@@_instruction_of_type:nnn \c_false_bool { Vdots }
4997              { #1 , down = #2 , up = #3 , middle = #4 }
4998          }
4999        }
5000      \bool_if:NF \l_@@_nullify_dots_bool
5001        { \phantom { \ensuremath { \@@_old_vdots } } } }
5002      \bool_gset_true:N \g_@@_empty_cell_bool
5003    }


5004    \cs_new_protected:Npn \@@_Ddots
5005      { \@@_collect_options:n { \@@_Ddots_i } }
5006    \exp_args:NNo \NewDocumentCommand \@@_Ddots_i \l_@@_argspec_tl
5007      {
5008        \int_case:nnF \c@iRow
5009          {
5010            0                 { \@@_error:nn { in~first~row } \Ddots }
5011            \l_@@_last_row_int { \@@_error:nn { in~last~row } \Ddots }
5012          }
5013          {
5014            \int_case:nnF \c@jCol
5015              {
5016                0                 { \@@_error:nn { in~first~col } \Ddots }
5017                \l_@@_last_col_int { \@@_error:nn { in~last~col } \Ddots }
5018              }
5019              {
5020                \keys_set_known:nn { NiceMatrix / Ddots } { #1 }
5021                \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Ddots }
5022                  { #1 , down = #2 , up = #3 , middle = #4 }
5023              }

5024
5025          }
5026      \bool_if:NF \l_@@_nullify_dots_bool
5027        { \phantom { \ensuremath { \@@_old_ddots } } }
5028      \bool_gset_true:N \g_@@_empty_cell_bool
5029    }


5030    \cs_new_protected:Npn \@@_Iddots
5031      { \@@_collect_options:n { \@@_Iddots_i } }
5032    \exp_args:NNo \NewDocumentCommand \@@_Iddots_i \l_@@_argspec_tl
5033      {
5034        \int_case:nnF \c@iRow
5035          {
5036            0                 { \@@_error:nn { in~first~row } \Iddots }
5037            \l_@@_last_row_int { \@@_error:nn { in~last~row } \Iddots }
5038          }
5039          {
5040            \int_case:nnF \c@jCol
5041              {
5042                0                 { \@@_error:nn { in~first~col } \Iddots }
5043                \l_@@_last_col_int { \@@_error:nn { in~last~col } \Iddots }
5044              }
5045              {
5046                \keys_set_known:nn { NiceMatrix / Ddots } { #1 }
5047                \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Iddots }
5048                  { #1 , down = #2 , up = #3 , middle = #4 }
5049              }
5050          }
5051      \bool_if:NF \l_@@_nullify_dots_bool
5052        { \phantom { \ensuremath { \@@_old_iddots } } }
```

```
5053        \bool_gset_true:N \g_@@_empty_cell_bool
5054      }
5055  }
```

End of the `\AddToHook`.

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

```
5056 \keys_define:nn { NiceMatrix / Ddots }
5057   {
5058     draw-first .bool_set:N = \l_@@_draw_first_bool ,
5059     draw-first .default:n = true ,
5060     draw-first .value_forbidden:n = true
5061   }
```

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

```
5062 \cs_new_protected:Npn \@@_Hspace:
5063   {
5064    \bool_gset_true:N \g_@@_empty_cell_bool
5065    \hspace
5066   }
```

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`.

```
5067 \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`.

```
5068 \cs_new:Npn \@@_Hdotsfor:
5069   {
5070     \bool_lazy_and:nnTF
5071       { \int_if_zero_p:n \c@jCol }
5072       { \int_if_zero_p:n \l_@@_first_col_int }
5073       {
5074         \bool_if:NTF \g_@@_after_col_zero_bool
5075           {
5076             \multicolumn { 1 } { c } { }
5077             \@@_Hdotsfor_i
5078           }
5079           { \@@_fatal:n { Hdotsfor~in~col~0 } }
5080       }
5081       {
5082         \multicolumn { 1 } { c } { }
5083         \@@_Hdotsfor_i
5084       }
5085   }
```

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:`).

```
5086 \hook_gput_code:nnn { begindocument } { . }
5087   {
5088     \cs_set_nopar:Npn \l_@@_argspec_tl { m m O { } E { _ ^ : } { { } { } { } } }
5089     \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.

```
5090     \cs_new_protected:Npn \@@_Hdotsfor_i
5091       { \@@_collect_options:n { \@@_Hdotsfor_ii } }
5092     \exp_args:NNo \NewDocumentCommand \@@_Hdotsfor_ii \l_@@_argspec_tl
5093       {
```

121

```
5094        \tl_gput_right:Nx \g_@@_HVdotsfor_lines_tl
5095          {
5096            \@@_Hdotsfor:nnnn
5097              { \int_use:N \c@iRow }
5098              { \int_use:N \c@jCol }
5099              { #2 }
5100              {
5101                #1 , #3 ,
5102                down = \exp_not:n { #4 } ,
5103                up = \exp_not:n { #5 } ,
5104                middle = \exp_not:n { #6 }
5105              }
5106          }
5107        \prg_replicate:nn { #2 - 1 }
5108          {
5109            &
5110            \multicolumn { 1 } { c } { }
5111            \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
5112          }
5113      }
5114   }

5115 \cs_new_protected:Npn \@@_Hdotsfor:nnnn #1 #2 #3 #4
5116   {
5117     \bool_set_false:N \l_@@_initial_open_bool
5118     \bool_set_false:N \l_@@_final_open_bool
```

For the row, it's easy.

```
5119     \int_set:Nn \l_@@_initial_i_int { #1 }
5120     \int_set_eq:NN \l_@@_final_i_int \l_@@_initial_i_int
```

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

```
5121     \int_compare:nNnTF { #2 } = \c_one_int
5122       {
5123         \int_set_eq:NN \l_@@_initial_j_int \c_one_int
5124         \bool_set_true:N \l_@@_initial_open_bool
5125       }
5126       {
5127         \cs_if_exist:cTF
5128           {
5129             pgf @ sh @ ns @ \@@_env:
5130             - \int_use:N \l_@@_initial_i_int
5131             - \int_eval:n { #2 - 1 }
5132           }
5133           { \int_set:Nn \l_@@_initial_j_int { #2 - 1 } }
5134           {
5135             \int_set:Nn \l_@@_initial_j_int { #2 }
5136             \bool_set_true:N \l_@@_initial_open_bool
5137           }
5138       }
5139     \int_compare:nNnTF { #2 + #3 -1 } = \c@jCol
5140       {
5141         \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5142         \bool_set_true:N \l_@@_final_open_bool
5143       }
5144       {
5145         \cs_if_exist:cTF
5146           {
5147             pgf @ sh @ ns @ \@@_env:
5148             - \int_use:N \l_@@_final_i_int
5149             - \int_eval:n { #2 + #3 }
5150           }
5151           { \int_set:Nn \l_@@_final_j_int { #2 + #3 } }
5152           {
```

```
5153              \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
5154              \bool_set_true:N \l_@@_final_open_bool
5155          }
5156        }
5157      \group_begin:
5158      \@@_open_shorten:
5159      \int_if_zero:nTF { #1 }
5160        { \color { nicematrix-first-row } }
5161        {
5162          \int_compare:nNnT { #1 } = \g_@@_row_total_int
5163            { \color { nicematrix-last-row } }
5164        }
5165
5166      \keys_set:nn { NiceMatrix / xdots } { #4 }
5167      \tl_if_empty:oF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
5168      \@@_actually_draw_Ldots:
5169      \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).

```
5170      \int_step_inline:nnn { #2 } { #2 + #3 - 1 }
5171        { \cs_set:cpn { @@ _ dotted _ #1 - ##1 } { } }
5172  }


5173 \hook_gput_code:nnn { begindocument } { . }
5174  {
5175    \cs_set_nopar:Npn \l_@@_argspec_tl { m m O { } E { _ ^ : } { { } { } { } } }
5176    \tl_set_rescan:Nno  \l_@@_argspec_tl { } \l_@@_argspec_tl
5177    \cs_new_protected:Npn \@@_Vdotsfor:
5178      { \@@_collect_options:n { \@@_Vdotsfor_i } }
5179    \exp_args:NNo \NewDocumentCommand \@@_Vdotsfor_i \l_@@_argspec_tl
5180      {
5181        \bool_gset_true:N \g_@@_empty_cell_bool
5182        \tl_gput_right:Nx \g_@@_HVdotsfor_lines_tl
5183          {
5184            \@@_Vdotsfor:nnnn
5185              { \int_use:N \c@iRow }
5186              { \int_use:N \c@jCol }
5187              { #2 }
5188              {
5189                #1 , #3 ,
5190                down = \exp_not:n { #4 } ,
5191                up = \exp_not:n { #5 } ,
5192                middle = \exp_not:n { #6 }
5193              }
5194          }
5195      }
5196  }


5197 \cs_new_protected:Npn \@@_Vdotsfor:nnnn #1 #2 #3 #4
5198  {
5199    \bool_set_false:N \l_@@_initial_open_bool
5200    \bool_set_false:N \l_@@_final_open_bool
```

For the column, it's easy.

```
5201      \int_set:Nn \l_@@_initial_j_int { #2 }
5202      \int_set_eq:NN \l_@@_final_j_int \l_@@_initial_j_int
```

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

```
5203       \int_compare:nNnTF { #1 } = \c_one_int
5204         {
5205           \int_set_eq:NN \l_@@_initial_i_int \c_one_int
5206           \bool_set_true:N \l_@@_initial_open_bool
5207         }
5208         {
5209           \cs_if_exist:cTF
5210             {
5211               pgf @ sh @ ns @ \@@_env:
5212               - \int_eval:n { #1 - 1 }
5213               - \int_use:N \l_@@_initial_j_int
5214             }
5215             { \int_set:Nn \l_@@_initial_i_int { #1 - 1 } }
5216             {
5217               \int_set:Nn \l_@@_initial_i_int { #1 }
5218               \bool_set_true:N \l_@@_initial_open_bool
5219             }
5220         }
5221       \int_compare:nNnTF { #1 + #3 -1 } = \c@iRow
5222         {
5223           \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5224           \bool_set_true:N \l_@@_final_open_bool
5225         }
5226         {
5227           \cs_if_exist:cTF
5228             {
5229               pgf @ sh @ ns @ \@@_env:
5230               - \int_eval:n { #1 + #3 }
5231               - \int_use:N \l_@@_final_j_int
5232             }
5233             { \int_set:Nn \l_@@_final_i_int { #1 + #3 } }
5234             {
5235               \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
5236               \bool_set_true:N \l_@@_final_open_bool
5237             }
5238         }
5239       \group_begin:
5240       \@@_open_shorten:
5241       \int_if_zero:nTF { #2 }
5242         { \color { nicematrix-first-col } }
5243         {
5244           \int_compare:nNnT { #2 } = \g_@@_col_total_int
5245             { \color { nicematrix-last-col } }
5246         }
5247       \keys_set:nn { NiceMatrix / xdots } { #4 }
5248       \tl_if_empty:oF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
5249       \@@_actually_draw_Vdots:
5250       \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).

```
5251       \int_step_inline:nnn { #1 } { #1 + #3 - 1 }
5252         { \cs_set:cpn { @@ _ dotted _ ##1 - #2 } { } }
5253   }
```

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

```
5254 \NewDocumentCommand \@@_rotate: { O { } }
5255   {
```

124

```
5256      \peek_remove_spaces:n
5257        {
5258          \bool_gset_true:N \g_@@_rotate_bool
5259          \keys_set:nn { NiceMatrix / rotate } { #1 }
5260        }
5261    }


5262 \keys_define:nn { NiceMatrix / rotate }
5263    {
5264      c .code:n = \bool_gset_true:N \g_@@_rotate_c_bool ,
5265      c .value_forbidden:n = true ,
5266      unknown .code:n = \@@_error:n { Unknown~key~for~rotate }
5267    }
```

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]

```
5268 \cs_new:Npn \@@_double_int_eval:n #1-#2 \q_stop
5269    {
5270      \tl_if_empty:nTF { #2 }
5271        { #1 }
5272        { \@@_double_int_eval_i:n #1-#2 \q_stop }
5273    }
5274 \cs_new:Npn \@@_double_int_eval_i:n #1-#2- \q_stop
5275    { \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).

```
5276 \hook_gput_code:nnn { begindocument } { . }
5277    {
5278      \cs_set_nopar:Npn \l_@@_argspec_tl
5279        { O { } m m ! O { } E { _ ^ : } { { } { } { } } }
5280      \tl_set_rescan:Nno  \l_@@_argspec_tl { } \l_@@_argspec_tl
5281      \exp_args:NNo \NewDocumentCommand \@@_line \l_@@_argspec_tl
5282        {
5283          \group_begin:
5284          \keys_set:nn { NiceMatrix / xdots } { #1 , #4 , down = #5 , up = #6 }
5285          \tl_if_empty:oF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
5286            \use:e
5287              {
5288                \@@_line_i:nn
5289                  { \@@_double_int_eval:n #2 - \q_stop }
5290                  { \@@_double_int_eval:n #3 - \q_stop }
```

---

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

```
5291                    }
5292              \group_end:
5293          }
5294      }
5295  \cs_new_protected:Npn \@@_line_i:nn #1 #2
5296      {
5297        \bool_set_false:N \l_@@_initial_open_bool
5298        \bool_set_false:N \l_@@_final_open_bool
5299        \bool_lazy_or:nnTF
5300          { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #1 } }
5301          { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #2 } }
5302          { \@@_error:nnn { unknown~cell~for~line~in~CodeAfter } { #1 } { #2 } }
```

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

```
5303          { \legacy_if:nF { measuring@ } { \@@_draw_line_ii:nn { #1 } { #2 } } }
5304      }
5305  \hook_gput_code:nnn { begindocument } { . }
5306      {
5307        \cs_new_protected:Npx \@@_draw_line_ii:nn #1 #2
5308          {
```

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:`.

```
5309              \c_@@_pgfortikzpicture_tl
5310              \@@_draw_line_iii:nn { #1 } { #2 }
5311              \c_@@_endpgfortikzpicture_tl
5312          }
5313      }
```

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

```
5314  \cs_new_protected:Npn \@@_draw_line_iii:nn #1 #2
5315      {
5316        \pgfrememberpicturepositiononpagetrue
5317        \pgfpointshapeborder { \@@_env: - #1 } { \@@_qpoint:n { #2 } }
5318        \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
5319        \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
5320        \pgfpointshapeborder { \@@_env: - #2 } { \@@_qpoint:n { #1 } }
5321        \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
5322        \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
5323        \@@_draw_line:
5324      }
```

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

```
5325 \cs_new:Npn \@@_if_row_less_than:nn #1 #2
5326   { \int_compare:nNnT { \c@iRow } < { #1 } { #2 } }
```

\@@_put_in_row_style will be used several times by \RowStyle.

```
5327 \cs_set_protected:Npn \@@_put_in_row_style:n #1
5328   {
5329     \tl_gput_right:Nx \g_@@_row_style_tl
5330       {
```

Be careful, \exp_not:N \@@_if_row_less_than:nn can't be replaced by a protected version of \@@_if_row_less_than:nn.

```
5331         \exp_not:N
5332         \@@_if_row_less_than:nn
5333           { \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).

```
5334         { \exp_not:n { #1 } \scan_stop: }
5335       }
5336   }
5337 \cs_generate_variant:Nn \@@_put_in_row_style:n { e }

5338 \keys_define:nn { NiceMatrix / RowStyle }
5339   {
5340     cell-space-top-limit .dim_set:N = \l_tmpa_dim ,
5341     cell-space-top-limit .value_required:n = true ,
5342     cell-space-bottom-limit .dim_set:N = \l_tmpb_dim ,
5343     cell-space-bottom-limit .value_required:n = true ,
5344     cell-space-limits .meta:n =
5345       {
5346         cell-space-top-limit = #1 ,
5347         cell-space-bottom-limit = #1 ,
5348       } ,
5349     color .tl_set:N = \l_@@_color_tl ,
5350     color .value_required:n = true ,
5351     bold .bool_set:N = \l_@@_bold_row_style_bool ,
5352     bold .default:n = true ,
5353     nb-rows .code:n =
5354       \str_if_eq:nnTF { #1 } { * }
5355         { \int_set:Nn \l_@@_key_nb_rows_int { 500 } }
5356         { \int_set:Nn \l_@@_key_nb_rows_int { #1 } } ,
5357     nb-rows .value_required:n = true ,
5358     rowcolor .tl_set:N = \l_tmpa_tl ,
5359     rowcolor .value_required:n = true ,
5360     unknown .code:n = \@@_error:n { Unknown~key~for~RowStyle }
5361   }

5362 \NewDocumentCommand \@@_RowStyle:n { O { } m }
5363   {
5364     \group_begin:
5365     \tl_clear:N \l_tmpa_tl
5366     \tl_clear:N \l_@@_color_tl
5367     \int_set_eq:NN \l_@@_key_nb_rows_int \c_one_int
5368     \dim_zero:N \l_tmpa_dim
5369     \dim_zero:N \l_tmpb_dim
5370     \keys_set:nn { NiceMatrix / RowStyle } { #1 }
```

If the key rowcolor has been used.

```
5371     \tl_if_empty:NF \l_tmpa_tl
5372       {
```

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

```
5373          \tl_gput_right:Nx \g_@@_pre_code_before_tl
5374            {
```

The command \@@_exp_color_arg:No is *fully expandable*.

```
5375              \@@_exp_color_arg:No \@@_rectanglecolor \l_tmpa_tl
5376                { \int_use:N \c@iRow - \int_use:N \c@jCol }
5377                { \int_use:N \c@iRow - * }
5378            }
```

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

```
5379          \int_compare:nNnT \l_@@_key_nb_rows_int > \c_one_int
5380            {
5381              \tl_gput_right:Nx \g_@@_pre_code_before_tl
5382                {
5383                  \@@_exp_color_arg:No \@@_rowcolor \l_tmpa_tl
5384                    {
5385                      \int_eval:n { \c@iRow + 1 }
5386                      - \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int - 1 }
5387                    }
5388                }
5389            }
5390        }
5391      \@@_put_in_row_style:n { \exp_not:n { #2 } }
```

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

```
5392      \dim_compare:nNnT \l_tmpa_dim > \c_zero_dim
5393        {
5394          \exp_args:Nx \@@_put_in_row_style:n
5395            {
5396              \tl_gput_right:Nn \exp_not:N \g_@@_cell_after_hook_tl
5397                {
```

It's not possible to chanage the following code by using \dim_set_eq:NN (because of expansion).

```
5398                  \dim_set:Nn \l_@@_cell_space_top_limit_dim
5399                    { \dim_use:N \l_tmpa_dim }
5400                }
5401            }
5402        }
```

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

```
5403      \dim_compare:nNnT \l_tmpb_dim > \c_zero_dim
5404        {
5405          \exp_args:Nx \@@_put_in_row_style:n
5406            {
5407              \tl_gput_right:Nn \exp_not:N \g_@@_cell_after_hook_tl
5408                {
5409                  \dim_set:Nn \l_@@_cell_space_bottom_limit_dim
5410                    { \dim_use:N \l_tmpb_dim }
5411                }
5412            }
5413        }
```

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

```
5414      \tl_if_empty:NF \l_@@_color_tl
5415        {
5416          \@@_put_in_row_style:e
5417            {
5418              \mode_leave_vertical:
5419              \@@_color:n { \l_@@_color_tl }
5420            }
5421        }
```

`\l_@@_bold_row_style_bool` is the value of the key `bold`.

```
5422      \bool_if:NT \l_@@_bold_row_style_bool
5423        {
5424          \@@_put_in_row_style:n
5425            {
5426              \exp_not:n
5427                {
5428                  \if_mode_math:
5429                    \c_math_toggle_token
5430                    \bfseries \boldmath
5431                    \c_math_toggle_token
5432                  \else:
5433                    \bfseries \boldmath
5434                  \fi:
5435                }
5436            }
5437        }
5438      \group_end:
5439      \g_@@_row_style_tl
5440      \ignorespaces
5441    }
```

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).

```
5442  \cs_new_protected:Npn \@@_add_to_colors_seq:nn #1 #2
5443    {
```

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.

```
5444      \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.

```
5445      \str_if_in:nnF { #1 } { !! }
5446        {
5447          \seq_map_indexed_inline:Nn \g_@@_colors_seq
5448            { \tl_if_eq:nnT { #1 } { ##2 } { \int_set:Nn \l_tmpa_int { ##1 } } }
5449        }
5450      \int_if_zero:nTF \l_tmpa_int
```

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

```
5451        {
5452          \seq_gput_right:Nn \g_@@_colors_seq { #1 }
5453          \tl_gset:cx { g_@@_color _ \seq_count:N \g_@@_colors_seq _ tl } { #2 }
5454        }
```

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

```
5455        { \tl_gput_right:cx { g_@@_color _ \int_use:N \l_tmpa_int _tl } { #2 } }
5456      }
```

```
5457  \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e n }
5458  \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e e }
```

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

```
5459  \cs_new_protected:Npn \@@_clip_with_rounded_corners:
5460    {
5461      \dim_compare:nNnT \l_@@_tab_rounded_corners_dim > \c_zero_dim
5462        {
```

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

```
5463          \group_begin:
5464          \pgfsetcornersarced
5465            {
5466              \pgfpoint
5467                { \l_@@_tab_rounded_corners_dim }
5468                { \l_@@_tab_rounded_corners_dim }
5469            }
```

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.

```
5470          \bool_if:NTF \l_@@_hvlines_bool
5471            {
5472              \pgfpathrectanglecorners
5473                {
5474                  \pgfpointadd
5475                    { \@@_qpoint:n { row-1 } }
5476                    { \pgfpoint { 0.5 \arrayrulewidth } { \c_zero_dim } }
5477                }
5478                {
5479                  \pgfpointadd
5480                    {
5481                      \@@_qpoint:n
5482                        { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
5483                    }
5484                    { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
5485                }
5486            }
5487            {
5488              \pgfpathrectanglecorners
5489                { \@@_qpoint:n { row-1 } }
5490                {
5491                  \pgfpointadd
5492                    {
5493                      \@@_qpoint:n
5494                        { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
5495                    }
5496                    { \pgfpoint \c_zero_dim \arrayrulewidth }
5497                }
5498            }
5499          \pgfusepath { clip }
5500          \group_end:
```

The TeX group was for \pgfsetcornersarced.

```
5501        }
5502    }
```

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).

```
5503 \cs_new_protected:Npn \@@_actually_color:
5504    {
5505      \pgfpicture
5506      \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.

```
5507      \@@_clip_with_rounded_corners:
5508      \seq_map_indexed_inline:Nn \g_@@_colors_seq
5509        {
5510          \int_compare:nNnTF { ##1 } = \c_one_int
5511            {
5512              \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_nocolor:n
5513              \use:c { g_@@_color _ 1 _tl }
5514              \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_normal:n
5515            }
5516            {
5517              \begin { pgfscope }
5518                \@@_color_opacity ##2
5519                \use:c { g_@@_color _ ##1 _tl }
5520                \tl_gclear:c { g_@@_color _ ##1 _tl }
5521                \pgfusepath { fill }
5522              \end { pgfscope }
5523            }
5524        }
5525      \endpgfpicture
5526    }
```

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

```
5527 \cs_new_protected:Npn \@@_color_opacity
5528    {
5529      \peek_meaning:NTF [
5530        { \@@_color_opacity:w }
5531        { \@@_color_opacity:w [ ] }
5532    }
```

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.

```
5533 \cs_new_protected:Npn \@@_color_opacity:w [ #1 ]
5534    {
5535      \tl_clear:N \l_tmpa_tl
5536      \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.

```
5537      \tl_if_empty:NF \l_tmpa_tl { \exp_args:No \pgfsetfillopacity \l_tmpa_tl }
5538      \tl_if_empty:NTF \l_tmpb_tl
5539        { \@declaredcolor }
5540        { \use:e { \exp_not:N \@undeclaredcolor [ \l_tmpb_tl ] } }
5541    }
```

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

```
5542 \keys_define:nn { nicematrix / color-opacity }
5543    {
```

```
5544        opacity .tl_set:N          = \l_tmpa_tl ,
5545        opacity .value_required:n = true
5546      }


5547  \cs_new_protected:Npn \@@_cartesian_color:nn #1 #2
5548    {
5549      \cs_set_nopar:Npn \l_@@_rows_tl { #1 }
5550      \cs_set_nopar:Npn \l_@@_cols_tl { #2 }
5551      \@@_cartesian_path:
5552    }
```

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

```
5553  \NewDocumentCommand \@@_rowcolor { O { } m m }
5554    {
5555      \tl_if_blank:nF { #2 }
5556        {
5557          \@@_add_to_colors_seq:en
5558            { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5559            { \@@_cartesian_color:nn { #3 } { - } }
5560        }
5561    }
```

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

```
5562  \NewDocumentCommand \@@_columncolor { O { } m m }
5563    {
5564      \tl_if_blank:nF { #2 }
5565        {
5566          \@@_add_to_colors_seq:en
5567            { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5568            { \@@_cartesian_color:nn { - } { #3 } }
5569        }
5570    }
```

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

```
5571  \NewDocumentCommand \@@_rectanglecolor { O { } m m m }
5572    {
5573      \tl_if_blank:nF { #2 }
5574        {
5575          \@@_add_to_colors_seq:en
5576            { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5577            { \@@_rectanglecolor:nnn { #3 } { #4 } { \c_zero_dim } }
5578        }
5579    }
```

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

```
5580  \NewDocumentCommand \@@_roundedrectanglecolor { O { } m m m m }
5581    {
5582      \tl_if_blank:nF { #2 }
5583        {
5584          \@@_add_to_colors_seq:en
5585            { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
5586            { \@@_rectanglecolor:nnn { #3 } { #4 } { #5 } }
5587        }
5588    }
```

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

```
5589  \cs_new_protected:Npn \@@_rectanglecolor:nnn #1 #2 #3
5590    {
5591      \@@_cut_on_hyphen:w #1 \q_stop
```

```
5592        \tl_clear_new:N \l_@@_tmpc_tl
5593        \tl_clear_new:N \l_@@_tmpd_tl
5594        \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
5595        \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
5596        \@@_cut_on_hyphen:w #2 \q_stop
5597        \tl_set:Nx \l_@@_rows_tl { \l_@@_tmpc_tl - \l_tmpa_tl }
5598        \tl_set:Nx \l_@@_cols_tl { \l_@@_tmpd_tl - \l_tmpb_tl }
```

The command `\@@_cartesian_path:n` takes in two implicit arguments: `\l_@@_cols_tl` and `\l_@@_rows_tl`.

```
5599        \@@_cartesian_path:n { #3 }
5600      }
```

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

```
5601 \NewDocumentCommand \@@_cellcolor { O { } m m }
5602   {
5603     \clist_map_inline:nn { #3 }
5604       { \@@_rectanglecolor [ #1 ] { #2 } { ##1 } { ##1 } } }
5605   }
```

```
5606 \NewDocumentCommand \@@_chessboardcolors { O { } m m  }
5607   {
5608     \int_step_inline:nn \c@iRow
5609       {
5610         \int_step_inline:nn \c@jCol
5611           {
5612             \int_if_even:nTF { ####1 + ##1 }
5613               { \@@_cellcolor [ #1 ] { #2 } }
5614               { \@@_cellcolor [ #1 ] { #3 } }
5615             { ##1 - ####1 }
5616           }
5617       }
5618   }
```

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".

```
5619 \NewDocumentCommand \@@_arraycolor { O { } m }
5620   {
5621     \@@_rectanglecolor [ #1 ] { #2 }
5622       { 1 - 1 }
5623       { \int_use:N \c@iRow - \int_use:N \c@jCol }
5624   }
```

```
5625 \keys_define:nn { NiceMatrix / rowcolors }
5626   {
5627     respect-blocks .bool_set:N = \l_@@_respect_blocks_bool ,
5628     respect-blocks .default:n = true ,
5629     cols .tl_set:N = \l_@@_cols_tl ,
5630     restart .bool_set:N = \l_@@_rowcolors_restart_bool ,
5631     restart .default:n = true ,
5632     unknown .code:n = \@@_error:n { Unknown~key~for~rowcolors }
5633   }
```

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*.

```
5634  \NewDocumentCommand \@@_rowlistcolors { O { } m m O { } }
5635    {
```

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

```
5636      \group_begin:
5637      \seq_clear_new:N \l_@@_colors_seq
5638      \seq_set_split:Nnn \l_@@_colors_seq { , } { #3 }
5639      \tl_clear_new:N \l_@@_cols_tl
5640      \cs_set_nopar:Npn \l_@@_cols_tl { - }
5641      \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).

```
5642      \int_zero_new:N \l_@@_color_int
5643      \int_set_eq:NN \l_@@_color_int \c_one_int
5644      \bool_if:NT \l_@@_respect_blocks_bool
5645        {
```

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`).

```
5646        \seq_set_eq:NN \l_tmpb_seq \g_@@_pos_of_blocks_seq
5647        \seq_set_filter:NNn \l_tmpa_seq \l_tmpb_seq
5648          { \@@_not_in_exterior_p:nnnnn ##1 }
5649        }
5650      \pgfpicture
5651      \pgf@relevantforpicturesizefalse
```

#2 is the list of intervals of rows.

```
5652      \clist_map_inline:nn { #2 }
5653        {
5654        \cs_set_nopar:Npn \l_tmpa_tl { ##1 }
5655        \tl_if_in:NnTF \l_tmpa_tl { - }
5656          { \@@_cut_on_hyphen:w ##1 \q_stop }
5657          { \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.

```
5658        \int_set:Nn \l_tmpa_int \l_tmpa_tl
5659        \int_set:Nn \l_@@_color_int
5660          { \bool_if:NTF \l_@@_rowcolors_restart_bool 1 \l_tmpa_tl }
5661        \int_zero_new:N \l_@@_tmpc_int
5662        \int_set:Nn \l_@@_tmpc_int \l_tmpb_tl
5663        \int_do_until:nNnn \l_tmpa_int > \l_@@_tmpc_int
5664          {
```

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

```
5665            \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).

```
5666            \bool_if:NT \l_@@_respect_blocks_bool
5667              {
5668                \seq_set_filter:NNn \l_tmpb_seq \l_tmpa_seq
5669                  { \@@_intersect_our_row_p:nnnnn ####1 }
5670                \seq_map_inline:Nn \l_tmpb_seq { \@@_rowcolors_i:nnnnn ####1 }
```

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

```
5671              }
5672            \tl_set:No \l_@@_rows_tl
5673              { \int_use:N \l_tmpa_int - \int_use:N \l_tmpb_int }
```

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

```
5674            \tl_clear_new:N \l_@@_color_tl
5675            \tl_set:Nx \l_@@_color_tl
5676              {
```

134

```
5677                    \@@_color_index:n
5678                      {
5679                        \int_mod:nn
5680                          { \l_@@_color_int - 1 }
5681                          { \seq_count:N \l_@@_colors_seq }
5682                        + 1
5683                      }
5684                  }
5685              \tl_if_empty:NF \l_@@_color_tl
5686                {
5687                  \@@_add_to_colors_seq:ee
5688                    { \tl_if_blank:nF { #1 } { [ #1 ] } { \l_@@_color_tl } }
5689                    { \@@_cartesian_color:nn { \l_@@_rows_tl } { \l_@@_cols_tl } }
5690                }
5691              \int_incr:N \l_@@_color_int
5692              \int_set:Nn \l_tmpa_int { \l_tmpb_int + 1 }
5693          }
5694        }
5695      \endpgfpicture
5696      \group_end:
5697    }
```

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.

```
5698  \cs_new:Npn \@@_color_index:n #1
5699    {
5700      \str_if_eq:eeTF { \seq_item:Nn \l_@@_colors_seq { #1 } } { = }
5701        { \@@_color_index:n { #1 - 1 } }
5702        { \seq_item:Nn \l_@@_colors_seq { #1 } }
5703    }
```

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.

```
5704  \NewDocumentCommand \@@_rowcolors { O { } m m m }
5705    { \@@_rowlistcolors [ #1 ] { #2 } { { #3 } , { #4 } } }
```

The braces around `#3` and `#4` are mandatory.

```
5706  \cs_new_protected:Npn \@@_rowcolors_i:nnnnn #1 #2 #3 #4 #5
5707    {
5708      \int_compare:nNnT { #3 } > \l_tmpb_int
5709        { \int_set:Nn \l_tmpb_int { #3 } }
5710    }
```

```
5711  \prg_new_conditional:Nnn \@@_not_in_exterior:nnnnn p
5712    {
5713      \int_if_zero:nTF { #4 }
5714        \prg_return_false:
5715        {
5716          \int_compare:nNnTF { #2 } > \c@jCol
5717            \prg_return_false:
5718            \prg_return_true:
5719        }
5720    }
```

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

```
5721  \prg_new_conditional:Nnn \@@_intersect_our_row:nnnnn p
5722    {
5723      \int_compare:nNnTF { #1 } > \l_tmpa_int
5724        \prg_return_false:
```

```
5725        {
5726          \int_compare:nNnTF \l_tmpa_int > { #3 }
5727            \prg_return_false:
5728            \prg_return_true:
5729        }
5730      }
```

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).

```
5731  \cs_new_protected:Npn \@@_cartesian_path_normal:n #1
5732    {
5733      \dim_compare:nNnTF { #1 } = \c_zero_dim
5734        {
5735          \bool_if:NTF
5736          \l_@@_nocolor_used_bool
5737          \@@_cartesian_path_normal_ii:
5738            {
5739              \seq_if_empty:NTF \l_@@_corners_cells_seq
5740                { \@@_cartesian_path_normal_i:n { #1 } }
5741                \@@_cartesian_path_normal_ii:
5742            }
5743        }
5744        { \@@_cartesian_path_normal_i:n { #1 } }
5745    }
```

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.

```
5746  \cs_new_protected:Npn \@@_cartesian_path_normal_i:n #1
5747    {
5748      \pgfsetcornersarced { \pgfpoint { #1 } { #1 } }
```
We begin the loop over the columns.
```
5749      \clist_map_inline:Nn \l_@@_cols_tl
5750        {
5751          \cs_set_nopar:Npn \l_tmpa_tl { ##1 }
5752          \tl_if_in:NnTF \l_tmpa_tl { - }
5753            { \@@_cut_on_hyphen:w ##1 \q_stop }
5754            { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
5755          \tl_if_empty:NTF \l_tmpa_tl
5756            { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5757            {
5758              \tl_if_eq:NNT \l_tmpa_tl \c_@@_star_tl
5759                { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5760            }
5761          \tl_if_empty:NTF \l_tmpb_tl
5762            { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
5763            {
5764              \tl_if_eq:NNT \l_tmpb_tl \c_@@_star_tl
5765                { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
5766            }
5767          \int_compare:nNnT \l_tmpb_tl > \g_@@_col_total_int
5768            { \tl_set:No \l_tmpb_tl { \int_use:N \g_@@_col_total_int } }
```
\l_@@_tmpc_tl will contain the number of column.
```
5769          \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
5770          \@@_qpoint:n { col - \l_tmpa_tl }
5771          \int_compare:nNnTF \l_@@_first_col_int = \l_tmpa_tl
5772            { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
5773            { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
```

```
5774          \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 }  }
5775          \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }
```
We begin the loop over the rows.
```
5776          \clist_map_inline:Nn \l_@@_rows_tl
5777            {
5778              \cs_set_nopar:Npn \l_tmpa_tl { ####1 }
5779              \tl_if_in:NnTF \l_tmpa_tl { - }
5780                { \@@_cut_on_hyphen:w ####1 \q_stop }
5781                { \@@_cut_on_hyphen:w ####1 - ####1 \q_stop }
5782              \tl_if_empty:NTF \l_tmpa_tl
5783                { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5784                {
5785                  \tl_if_eq:NNT \l_tmpa_tl \c_@@_star_tl
5786                    { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5787                }
5788              \tl_if_empty:NTF \l_tmpb_tl
5789                { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }
5790                {
5791                  \tl_if_eq:NNT \l_tmpb_tl \c_@@_star_tl
5792                    { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }
5793                }
5794              \int_compare:nNnT \l_tmpb_tl > \g_@@_row_total_int
5795                { \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.
```
5796                  \cs_if_exist:cF
5797                    { @@ _ \l_tmpa_tl _ \l_@@_tmpc_tl _ nocolor }
5798                    {
5799                      \@@_qpoint:n { row - \int_eval:n { \l_tmpb_tl + 1 } }
5800                      \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
5801                      \@@_qpoint:n { row - \l_tmpa_tl }
5802                      \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
5803                      \pgfpathrectanglecorners
5804                        { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
5805                        { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5806                    }
5807                }
5808            }
5809        }
```
Now, the case where the cells will be colored cell by cell (it's mandatory for example if the key corners is used).
```
5810 \cs_new_protected:Npn \@@_cartesian_path_normal_ii:
5811    {
5812      \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
5813      \@@_expand_clist:NN \l_@@_rows_tl \c@iRow
```
We begin the loop over the columns.
```
5814        \clist_map_inline:Nn \l_@@_cols_tl
5815          {
5816            \@@_qpoint:n { col - ##1 }
5817            \int_compare:nNnTF \l_@@_first_col_int = { ##1 }
5818              { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
5819              { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
5820            \@@_qpoint:n { col - \int_eval:n { ##1 + 1 }  }
5821            \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }
```
We begin the loop over the rows.
```
5822          \clist_map_inline:Nn \l_@@_rows_tl
5823            {
5824              \seq_if_in:NnF \l_@@_corners_cells_seq
5825                { ####1 - ##1 }
5826                {
5827                  \@@_qpoint:n { row - \int_eval:n { ####1 + 1 } }
```

```
5828                \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
5829                \@@_qpoint:n { row - ####1 }
5830                \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
5831                \cs_if_exist:cF { @@ _ ####1 _ ##1 _ nocolor }
5832                  {
5833                    \pgfpathrectanglecorners
5834                      { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
5835                      { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5836                  }
5837              }
5838          }
5839        }
5840    }
```

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).

```
5841 \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

```
5842 \cs_new_protected:Npn \@@_cartesian_path_nocolor:n #1
5843   {
5844     \bool_set_true:N \l_@@_nocolor_used_bool
5845     \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
5846     \@@_expand_clist:NN \l_@@_rows_tl \c@iRow
```
We begin the loop over the columns.
```
5847     \clist_map_inline:Nn \l_@@_rows_tl
5848       {
5849         \clist_map_inline:Nn \l_@@_cols_tl
5850           { \cs_set:cpn { @@ _ ##1 _ ####1 _ nocolor } { } }
5851       }
5852   }
```

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.

```
5853 \cs_new_protected:Npn \@@_expand_clist:NN #1 #2
5854   {
5855     \clist_set_eq:NN \l_tmpa_clist #1
5856     \clist_clear:N #1
5857     \clist_map_inline:Nn \l_tmpa_clist
5858       {
5859         \cs_set_nopar:Npn \l_tmpa_tl { ##1 }
5860         \tl_if_in:NnTF \l_tmpa_tl { - }
5861           { \@@_cut_on_hyphen:w ##1 \q_stop }
5862           { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
5863         \bool_lazy_or:nnT
5864           { \tl_if_blank_p:o \l_tmpa_tl }
5865           { \str_if_eq_p:on \l_tmpa_tl { * } }
5866           { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
5867         \bool_lazy_or:nnT
5868           { \tl_if_blank_p:o \l_tmpb_tl }
5869           { \str_if_eq_p:on \l_tmpb_tl { * } }
5870           { \tl_set:No \l_tmpb_tl { \int_use:N #2 } }
5871         \int_compare:nNnT \l_tmpb_tl > #2
5872           { \tl_set:No \l_tmpb_tl { \int_use:N #2 } }
5873         \int_step_inline:nnn \l_tmpa_tl \l_tmpb_tl
5874           { \clist_put_right:Nn #1 { ####1 } }
5875       }
5876   }
```

138

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

```
5877 \NewDocumentCommand \@@_cellcolor_tabular { O { } m }
5878   {
5879     \@@_test_color_inside:
5880     \tl_gput_right:Nx \g_@@_pre_code_before_tl
5881       {
```

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).

```
5882         \@@_cellcolor [ #1 ] { \exp_not:n { #2 } }
5883           { \int_use:N \c@iRow - \int_use:N \c@jCol }
5884       }
5885     \ignorespaces
5886   }
```

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

```
5887 \NewDocumentCommand \@@_rowcolor_tabular { O { } m }
5888   {
5889     \@@_test_color_inside:
5890     \tl_gput_right:Nx \g_@@_pre_code_before_tl
5891       {
5892         \@@_rectanglecolor [ #1 ] { \exp_not:n { #2 } }
5893           { \int_use:N \c@iRow - \int_use:N \c@jCol }
5894           { \int_use:N \c@iRow - \exp_not:n { \int_use:N \c@jCol } }
5895       }
5896     \ignorespaces
5897   }
```

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).

```
5898 \NewDocumentCommand { \@@_rowcolors_tabular } { O { } m m }
5899   { \@@_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.

```
5900 \NewDocumentCommand { \@@_rowlistcolors_tabular } { O { } m O { } }
5901   {
5902     \@@_test_color_inside:
5903     \peek_remove_spaces:n
5904       { \@@_rowlistcolors_tabular:nnn { #1 } { #2 } { #3 } }
5905   }
```

```
5906 \cs_new_protected:Npn \@@_rowlistcolors_tabular:nnn #1 #2 #3
5907   {
```

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`.

```
5908     \seq_gclear:N \g_tmpa_seq
5909     \seq_map_inline:Nn \g_@@_rowlistcolors_seq
5910       { \@@_rowlistcolors_tabular_i:nnnn ##1 }
5911     \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`.

```
5912      \seq_gput_right:Nx \g_@@_rowlistcolors_seq
5913        {
5914          { \int_use:N \c@iRow }
5915          { \exp_not:n { #1 } }
5916          { \exp_not:n { #2 } }
5917          { restart , cols = \int_use:N \c@jCol - , \exp_not:n { #3 } }
5918        }
5919    }
```

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`).

```
5920 \cs_new_protected:Npn \@@_rowlistcolors_tabular_i:nnnn #1 #2 #3 #4
5921    {
5922      \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).

```
5923        { \seq_gput_right:Nn \g_tmpa_seq { { #1 } { #2 } { #3 } { #4 } } }
5924        {
5925          \tl_gput_right:Nx \g_@@_pre_code_before_tl
5926            {
5927              \@@_rowlistcolors
5928                [ \exp_not:n { #2 } ]
5929                { #1 - \int_eval:n { \c@iRow - 1 } }
5930                { \exp_not:n { #3 } }
5931                [ \exp_not:n { #4 } ]
5932            }
5933        }
5934    }
```

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.

```
5935 \cs_new_protected:Npn \@@_clear_rowlistcolors_seq:
5936    {
5937      \seq_map_inline:Nn \g_@@_rowlistcolors_seq
5938        { \@@_rowlistcolors_tabular_ii:nnnn ##1 }
5939      \seq_gclear:N \g_@@_rowlistcolors_seq
5940    }
```

```
5941 \cs_new_protected:Npn \@@_rowlistcolors_tabular_ii:nnnn #1 #2 #3 #4
5942    {
5943      \tl_gput_right:Nn \g_@@_pre_code_before_tl
5944        { \@@_rowlistcolors [ #2 ] { #1 } { #3 } [ #4 ] }
5945    }
```

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.

```
5946 \NewDocumentCommand \@@_columncolor_preamble { O { } m }
5947    {
```

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).

```
5948        \int_compare:nNnT \c@jCol > \g_@@_col_total_int
5949          {
```

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).

```
5950          \tl_gput_left:Nx \g_@@_pre_code_before_tl
5951            {
5952              \exp_not:N \columncolor [ #1 ]
5953                { \exp_not:n { #2 } } { \int_use:N \c@jCol }
5954            }
5955          }
5956    }
```

```
5957 \hook_gput_code:nnn { begindocument } { . }
5958   {
5959     \IfPackageLoadedTF { colortbl }
5960       {
5961         \cs_set_eq:NN \@@_old_cellcolor \cellcolor
5962         \cs_set_eq:NN \@@_old_rowcolor \rowcolor
5963         \cs_new_protected:Npn \@@_revert_colortbl:
5964           {
5965             \hook_gput_code:nnn { env / tabular / begin } { nicematrix }
5966               {
5967                 \cs_set_eq:NN \cellcolor \@@_old_cellcolor
5968                 \cs_set_eq:NN \rowcolor \@@_old_rowcolor
5969               }
5970           }
5971       }
5972       { \cs_new_protected:Npn \@@_revert_colortbl: { } }
5973   }
```

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`.

```
5974 \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`.

```
5975 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix:n #1
5976   {
5977     \int_if_zero:nTF \l_@@_first_col_int
5978       { \@@_OnlyMainNiceMatrix_i:n { #1 } }
5979       {
5980         \int_if_zero:nTF \c@jCol
```

```
5981              {
5982                \int_compare:nNnF \c@iRow = { -1 }
5983                  { \int_compare:nNnF \c@iRow = { \l_@@_last_row_int - 1 } { #1 } }
5984              }
5985            { \@@_OnlyMainNiceMatrix_i:n { #1 } }
5986        }
5987    }
```

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.

```
5988  \cs_new_protected:Npn \@@_OnlyMainNiceMatrix_i:n #1
5989    {
5990      \int_if_zero:nF \c@iRow
5991        {
5992          \int_compare:nNnF \c@iRow = \l_@@_last_row_int
5993            {
5994              \int_compare:nNnT \c@jCol > \c_zero_int
5995                { \bool_if:NF \l_@@_in_last_col_bool { #1 } }
5996            }
5997        }
5998    }
```

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.

```
5999  \keys_define:nn { NiceMatrix / Rules }
6000    {
6001      position .int_set:N = \l_@@_position_int ,
6002      position .value_required:n = true ,
6003      start .int_set:N = \l_@@_start_int ,
6004      end .code:n =
6005        \bool_lazy_or:nnTF
6006          { \tl_if_empty_p:n { #1 } }
6007          { \str_if_eq_p:nn { #1 } { last } }
6008          { \int_set_eq:NN \l_@@_end_int \c@jCol }
6009          { \int_set:Nn \l_@@_end_int { #1 } }
6010    }
```

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.

```
6011  \keys_define:nn { NiceMatrix / RulesBis }
6012    {
6013      multiplicity .int_set:N = \l_@@_multiplicity_int ,
6014      multiplicity .initial:n = 1 ,
6015      dotted .bool_set:N = \l_@@_dotted_bool ,
6016      dotted .initial:n = false ,
6017      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]`.

```
6018    color .code:n =
6019      \@@_set_CT@arc@:n { #1 }
6020      \tl_set:Nn \l_@@_rule_color_tl { #1 } ,
6021    color .value_required:n = true ,
6022    sep-color .code:n = \@@_set_CT@drsc@:n { #1 } ,
6023    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.

```
6024    tikz .code:n =
6025      \IfPackageLoadedTF { tikz }
6026        { \clist_put_right:Nn \l_@@_tikz_rule_tl { #1 } }
6027        { \@@_error:n { tikz~without~tikz } } ,
6028    tikz .value_required:n = true ,
6029    total-width .dim_set:N = \l_@@_rule_width_dim ,
6030    total-width .value_required:n = true ,
6031    width .meta:n = { total-width = #1 } ,
6032    unknown .code:n = \@@_error:n { Unknow~key~for~RulesBis }
6033  }
```

The vertical rules

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

```
6034 \cs_new_protected:Npn \@@_vline:n #1
6035  {
```

The group is for the options.

```
6036    \group_begin:
6037    \int_set_eq:NN \l_@@_end_int \c@iRow
6038    \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).

```
6039    \int_compare:nNnT \l_@@_position_int < { \c@jCol + 2 }
6040      \@@_vline_i:
6041    \group_end:
6042  }

6043 \cs_new_protected:Npn \@@_vline_i:
6044  {
```

`\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`.

```
6045    \tl_set:No \l_tmpb_tl { \int_use:N \l_@@_position_int }
6046    \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
6047      \l_tmpa_tl
6048      {
```

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.

```
6049        \bool_gset_true:N \g_tmpa_bool
6050        \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6051          { \@@_test_vline_in_block:nnnnn ##1 }
6052        \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
6053          { \@@_test_vline_in_block:nnnnn ##1 }
6054        \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
```

```
6055                { \@@_test_vline_in_stroken_block:nnnn ##1 }
6056            \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_v:
6057            \bool_if:NTF \g_tmpa_bool
6058              {
6059                \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.

```
6060                  { \int_set:Nn \l_@@_local_start_int \l_tmpa_tl }
6061              }
6062              {
6063                \int_compare:nNnT \l_@@_local_start_int > \c_zero_int
6064                  {
6065                    \int_set:Nn \l_@@_local_end_int { \l_tmpa_tl - 1 }
6066                    \@@_vline_ii:
6067                    \int_zero:N \l_@@_local_start_int
6068                  }
6069              }
6070          }
6071      \int_compare:nNnT \l_@@_local_start_int > \c_zero_int
6072        {
6073          \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
6074          \@@_vline_ii:
6075        }
6076    }


6077 \cs_new_protected:Npn \@@_test_in_corner_v:
6078    {
6079      \int_compare:nNnTF \l_tmpb_tl = { \int_eval:n { \c@jCol + 1 } }
6080        {
6081          \seq_if_in:NxT
6082            \l_@@_corners_cells_seq
6083            { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
6084            { \bool_set_false:N \g_tmpa_bool }
6085        }
6086        {
6087          \seq_if_in:NxT
6088            \l_@@_corners_cells_seq
6089            { \l_tmpa_tl - \l_tmpb_tl }
6090            {
6091              \int_compare:nNnTF \l_tmpb_tl = \c_one_int
6092                { \bool_set_false:N \g_tmpa_bool }
6093                {
6094                  \seq_if_in:NxT
6095                    \l_@@_corners_cells_seq
6096                    { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
6097                    { \bool_set_false:N \g_tmpa_bool }
6098                }
6099            }
6100        }
6101    }


6102 \cs_new_protected:Npn \@@_vline_ii:
6103    {
6104      \tl_clear:N \l_@@_tikz_rule_tl
6105      \keys_set:nV { NiceMatrix / RulesBis } \l_@@_other_keys_tl
6106      \bool_if:NTF \l_@@_dotted_bool
6107        \@@_vline_iv:
6108        {
6109          \tl_if_empty:NTF \l_@@_tikz_rule_tl
6110            \@@_vline_iii:
6111            \@@_vline_v:
```

144

```
6112              }
6113        }
```

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

```
6114  \cs_new_protected:Npn \@@_vline_iii:
6115    {
6116      \pgfpicture
6117      \pgfrememberpicturepositiononpagetrue
6118      \pgf@relevantforpicturesizefalse
6119      \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6120      \dim_set_eq:NN \l_tmpa_dim \pgf@y
6121      \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6122      \dim_set:Nn \l_tmpb_dim
6123        {
6124          \pgf@x
6125          - 0.5 \l_@@_rule_width_dim
6126          +
6127          ( \arrayrulewidth * \l_@@_multiplicity_int
6128            + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6129        }
6130      \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6131      \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
6132      \bool_lazy_all:nT
6133        {
6134          { \int_compare_p:nNn \l_@@_multiplicity_int > \c_one_int }
6135          { \cs_if_exist_p:N \CT@drsc@ }
6136          { ! \tl_if_blank_p:o \CT@drsc@ }
6137        }
6138        {
6139          \group_begin:
6140          \CT@drsc@
6141          \dim_add:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }
6142          \dim_sub:Nn \l_@@_tmpc_dim { 0.5 \arrayrulewidth }
6143          \dim_set:Nn \l_@@_tmpd_dim
6144            {
6145              \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
6146              * ( \l_@@_multiplicity_int - 1 )
6147            }
6148          \pgfpathrectanglecorners
6149            { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6150            { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
6151          \pgfusepath { fill }
6152          \group_end:
6153        }
6154      \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6155      \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
6156      \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6157        {
6158          \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
6159          \dim_sub:Nn \l_tmpb_dim \doublerulesep
6160          \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6161          \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
6162        }
6163      \CT@arc@
6164      \pgfsetlinewidth { 1.1 \arrayrulewidth }
6165      \pgfsetrectcap
6166      \pgfusepathqstroke
6167      \endpgfpicture
6168    }
```

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

```
6169  \cs_new_protected:Npn \@@_vline_iv:
```

145

```
6170    {
6171      \pgfpicture
6172      \pgfrememberpicturepositiononpagetrue
6173      \pgf@relevantforpicturesizefalse
6174      \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6175      \dim_set:Nn \l_@@_x_initial_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
6176      \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
6177      \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6178      \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
6179      \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6180      \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
6181      \CT@arc@
6182      \@@_draw_line:
6183      \endpgfpicture
6184    }
```

The following code is for the case when the user uses the key `tikz`.

```
6185  \cs_new_protected:Npn \@@_vline_v:
6186    {
6187      \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.

```
6188      \CT@arc@
6189      \tl_if_empty:NF \l_@@_rule_color_tl
6190        { \tl_put_right:Nx \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6191      \pgfrememberpicturepositiononpagetrue
6192      \pgf@relevantforpicturesizefalse
6193      \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
6194      \dim_set_eq:NN \l_tmpa_dim \pgf@y
6195      \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
6196      \dim_set:Nn \l_tmpb_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
6197      \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
6198      \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
6199      \exp_args:No \tikzset \l_@@_tikz_rule_tl
6200      \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6201        ( \l_tmpb_dim , \l_tmpa_dim ) --
6202        ( \l_tmpb_dim , \l_@@_tmpc_dim ) ;
6203      \end { tikzpicture }
6204    }
```

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).

```
6205  \cs_new_protected:Npn \@@_draw_vlines:
6206    {
6207      \int_step_inline:nnn
6208        { \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool 2 1 }
6209        {
6210          \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool
6211            \c@jCol
6212            { \int_eval:n { \c@jCol + 1 } }
6213        }
6214        {
6215          \tl_if_eq:NNF \l_@@_vlines_clist \c_@@_all_tl
6216            { \clist_if_in:NnT \l_@@_vlines_clist { ##1 } }
6217            { \@@_vline:n { position = ##1 , total-width = \arrayrulewidth } }
6218        }
6219    }
```

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}.

```
6220 \cs_new_protected:Npn \@@_hline:n #1
6221   {
```

The group is for the options.

```
6222     \group_begin:
6223     \int_zero_new:N \l_@@_end_int
6224     \int_set_eq:NN \l_@@_end_int \c@jCol
6225     \keys_set_known:nnN { NiceMatrix / Rules } { #1 } \l_@@_other_keys_tl
6226     \@@_hline_i:
6227     \group_end:
6228   }
6229 \cs_new_protected:Npn \@@_hline_i:
6230   {
6231     \int_zero_new:N \l_@@_local_start_int
6232     \int_zero_new:N \l_@@_local_end_int
```

\l_tmpa_tl is the number of row and \l_tmpb_tl the number of column. When we have found a column corresponding to a rule to draw, we note its number in \l_@@_tmpc_tl.

```
6233     \tl_set:No \l_tmpa_tl { \int_use:N \l_@@_position_int }
6234     \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
6235       \l_tmpb_tl
6236       {
```

The boolean \g_tmpa_bool indicates whether the small horizontal rule will be drawn. If we find that it is in a block (a real block, created by \Block or a virtual block corresponding to a dotted line, created by \Cdots, \Vdots, etc.), we will set \g_tmpa_bool to false and the small horizontal rule won't be drawn.

```
6237         \bool_gset_true:N \g_tmpa_bool
6238         \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6239           { \@@_test_hline_in_block:nnnnn ##1 }
6240         \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
6241           { \@@_test_hline_in_block:nnnnn ##1 }
6242         \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
6243           { \@@_test_hline_in_stroken_block:nnnn ##1 }
6244         \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_h:
6245         \bool_if:NTF \g_tmpa_bool
6246           {
6247             \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.

```
6248               { \int_set:Nn \l_@@_local_start_int \l_tmpb_tl }
6249           }
6250           {
6251             \int_compare:nNnT \l_@@_local_start_int > \c_zero_int
6252               {
6253                 \int_set:Nn \l_@@_local_end_int { \l_tmpb_tl - 1 }
6254                 \@@_hline_ii:
6255                 \int_zero:N \l_@@_local_start_int
6256               }
6257           }
6258       }
6259     \int_compare:nNnT \l_@@_local_start_int > \c_zero_int
6260       {
6261         \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
6262         \@@_hline_ii:
6263       }
6264   }
```

```
6265 \cs_new_protected:Npn \@@_test_in_corner_h:
6266   {
6267     \int_compare:nNnTF \l_tmpa_tl = { \int_eval:n { \c@iRow + 1 } }
6268       {
6269         \seq_if_in:NxT
6270           \l_@@_corners_cells_seq
6271           { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
6272           { \bool_set_false:N \g_tmpa_bool }
6273       }
6274       {
6275         \seq_if_in:NxT
6276           \l_@@_corners_cells_seq
6277           { \l_tmpa_tl - \l_tmpb_tl }
6278           {
6279             \int_compare:nNnTF \l_tmpa_tl = \c_one_int
6280               { \bool_set_false:N \g_tmpa_bool }
6281               {
6282                 \seq_if_in:NxT
6283                   \l_@@_corners_cells_seq
6284                   { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
6285                   { \bool_set_false:N \g_tmpa_bool }
6286               }
6287           }
6288       }
6289   }


6290 \cs_new_protected:Npn \@@_hline_ii:
6291   {
6292     \tl_clear:N \l_@@_tikz_rule_tl
6293     \keys_set:nV { NiceMatrix / RulesBis } \l_@@_other_keys_tl
6294     \bool_if:NTF \l_@@_dotted_bool
6295       \@@_hline_iv:
6296       {
6297         \tl_if_empty:NTF \l_@@_tikz_rule_tl
6298           \@@_hline_iii:
6299           \@@_hline_v:
6300       }
6301   }
```

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

```
6302 \cs_new_protected:Npn \@@_hline_iii:
6303   {
6304     \pgfpicture
6305     \pgfrememberpicturepositiononpagetrue
6306     \pgf@relevantforpicturesizefalse
6307     \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6308     \dim_set_eq:NN \l_tmpa_dim \pgf@x
6309     \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6310     \dim_set:Nn \l_tmpb_dim
6311       {
6312         \pgf@y
6313         - 0.5 \l_@@_rule_width_dim
6314         +
6315         ( \arrayrulewidth * \l_@@_multiplicity_int
6316             + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
6317       }
6318     \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6319     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
6320     \bool_lazy_all:nT
6321       {
6322         { \int_compare_p:nNn \l_@@_multiplicity_int > \c_one_int }
6323         { \cs_if_exist_p:N \CT@drsc@ }
```

```
6324        { ! \tl_if_blank_p:o \CT@drsc@ }
6325      }
6326      {
6327        \group_begin:
6328        \CT@drsc@
6329        \dim_set:Nn \l_@@_tmpd_dim
6330          {
6331            \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
6332            * ( \l_@@_multiplicity_int - 1 )
6333          }
6334        \pgfpathrectanglecorners
6335          { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6336          { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
6337        \pgfusepathqfill
6338        \group_end:
6339      }
6340      \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6341      \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
6342      \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
6343        {
6344          \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
6345          \dim_sub:Nn \l_tmpb_dim \doublerulesep
6346          \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
6347          \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
6348        }
6349      \CT@arc@
6350      \pgfsetlinewidth { 1.1 \arrayrulewidth }
6351      \pgfsetrectcap
6352      \pgfusepathqstroke
6353      \endpgfpicture
6354    }
```

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}
```

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ \hline 1 & 2 & 3 & 4 \\ \hdashline 1 & 2 & 3 & 4 \end{bmatrix}$$

```
6355 \cs_new_protected:Npn \@@_hline_iv:
6356   {
6357     \pgfpicture
6358     \pgfrememberpicturepositiononpagetrue
6359     \pgf@relevantforpicturesizefalse
6360     \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6361     \dim_set:Nn \l_@@_y_initial_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
6362     \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
6363     \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6364     \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
6365     \int_compare:nNnT \l_@@_local_start_int = \c_one_int
6366       {
```

```
6367        \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
6368        \bool_if:NF \g_@@_delims_bool
6369          { \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.

```
6370        \tl_if_eq:NnF \g_@@_left_delim_tl (
6371          { \dim_add:Nn \l_@@_x_initial_dim  { 0.5 \l_@@_xdots_inter_dim } }
6372        }
6373      \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6374      \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
6375      \int_compare:nNnT \l_@@_local_end_int = \c@jCol
6376        {
6377          \dim_add:Nn \l_@@_x_final_dim \l_@@_right_margin_dim
6378          \bool_if:NF \g_@@_delims_bool
6379            { \dim_add:Nn \l_@@_x_final_dim \arraycolsep }
6380          \tl_if_eq:NnF \g_@@_right_delim_tl )
6381            { \dim_gsub:Nn \l_@@_x_final_dim { 0.5 \l_@@_xdots_inter_dim } }
6382        }
6383      \CT@arc@
6384      \@@_draw_line:
6385      \endpgfpicture
6386    }
```

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`).

```
6387 \cs_new_protected:Npn \@@_hline_v:
6388    {
6389      \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.

```
6390      \CT@arc@
6391      \tl_if_empty:NF \l_@@_rule_color_tl
6392        { \tl_put_right:Nx \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
6393      \pgfrememberpicturepositiononpagetrue
6394      \pgf@relevantforpicturesizefalse
6395      \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
6396      \dim_set_eq:NN \l_tmpa_dim \pgf@x
6397      \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
6398      \dim_set:Nn \l_tmpb_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
6399      \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
6400      \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
6401      \exp_args:No \tikzset \l_@@_tikz_rule_tl
6402      \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
6403        ( \l_tmpa_dim , \l_tmpb_dim ) --
6404        ( \l_@@_tmpc_dim , \l_tmpb_dim ) ;
6405      \end { tikzpicture }
6406    }
```

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).

```
6407 \cs_new_protected:Npn \@@_draw_hlines:
6408    {
6409      \int_step_inline:nnn
6410        { \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool 2 1 }
6411        {
6412          \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool
6413            \c@iRow
6414            { \int_eval:n { \c@iRow + 1 } }
```

```
6415        }
6416        {
6417          \tl_if_eq:NNF \l_@@_hlines_clist \c_@@_all_tl
6418            { \clist_if_in:NnT \l_@@_hlines_clist { ##1 } }
6419            { \@@_hline:n { position = ##1 , total-width = \arrayrulewidth } }
6420        }
6421    }
```

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

```
6422 \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.

```
6423 \cs_set:Npn \@@_Hline_i:n #1
6424    {
6425      \peek_remove_spaces:n
6426        {
6427          \peek_meaning:NTF \Hline
6428            { \@@_Hline_ii:nn { #1 + 1 } }
6429            { \@@_Hline_iii:n { #1 } }
6430        }
6431    }
```

```
6432 \cs_set:Npn \@@_Hline_ii:nn #1 #2 { \@@_Hline_i:n { #1 } }
```

```
6433 \cs_set:Npn \@@_Hline_iii:n #1
6434    { \@@_collect_options:n { \@@_Hline_iv:nn { #1 } } }
```

```
6435 \cs_set:Npn \@@_Hline_iv:nn #1 #2
6436    {
6437      \@@_compute_rule_width:n { multiplicity = #1 , #2 }
6438      \skip_vertical:N \l_@@_rule_width_dim
6439      \tl_gput_right:Nx \g_@@_pre_code_after_tl
6440        {
6441          \@@_hline:n
6442            {
6443              multiplicity = #1 ,
6444              position = \int_eval:n { \c@iRow + 1 } ,
6445              total-width = \dim_use:N \l_@@_rule_width_dim ,
6446              #2
6447            }
6448        }
6449      \egroup
6450    }
```

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`).

```
6451 \cs_new_protected:Npn \@@_custom_line:n #1
6452    {
6453      \str_clear_new:N \l_@@_command_str
6454      \str_clear_new:N \l_@@_ccommand_str
6455      \str_clear_new:N \l_@@_letter_str
6456      \tl_clear_new:N \l_@@_other_keys_tl
6457      \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.

```
6458      \bool_lazy_all:nTF
6459        {
```

```
6460            { \str_if_empty_p:N \l_@@_letter_str }
6461            { \str_if_empty_p:N \l_@@_command_str }
6462            { \str_if_empty_p:N \l_@@_ccommand_str }
6463          }
6464          { \@@_error:n { No~letter~and~no~command } }
6465          { \exp_args:No \@@_custom_line_i:n \l_@@_other_keys_tl }
6466      }
6467  \keys_define:nn { NiceMatrix / custom-line }
6468      {
6469        letter .str_set:N = \l_@@_letter_str ,
6470        letter .value_required:n = true ,
6471        command .str_set:N = \l_@@_command_str ,
6472        command .value_required:n = true ,
6473        ccommand .str_set:N = \l_@@_ccommand_str ,
6474        ccommand .value_required:n = true ,
6475      }


6476  \cs_new_protected:Npn \@@_custom_line_i:n #1
6477      {
```

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

```
6478        \bool_set_false:N \l_@@_tikz_rule_bool
6479        \bool_set_false:N \l_@@_dotted_rule_bool
6480        \bool_set_false:N \l_@@_color_bool

6481        \keys_set:nn { NiceMatrix / custom-line-bis } { #1 }
6482        \bool_if:NT \l_@@_tikz_rule_bool
6483          {
6484            \IfPackageLoadedTF { tikz }
6485              { }
6486              { \@@_error:n { tikz~in~custom-line~without~tikz } }
6487            \bool_if:NT \l_@@_color_bool
6488              { \@@_error:n { color~in~custom-line~with~tikz } }
6489          }
6490        \bool_if:NT \l_@@_dotted_rule_bool
6491          {
6492            \int_compare:nNnT \l_@@_multiplicity_int > \c_one_int
6493              { \@@_error:n { key~multiplicity~with~dotted } }
6494          }
6495        \str_if_empty:NF \l_@@_letter_str
6496          {
6497            \int_compare:nTF { \str_count:N \l_@@_letter_str != 1 }
6498              { \@@_error:n { Several~letters } }
6499              {
6500                \exp_args:NnV \tl_if_in:NnTF
6501                  \c_@@_forbidden_letters_str \l_@@_letter_str
6502                  { \@@_error:ne { Forbidden~letter } \l_@@_letter_str }
6503                  {
```

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.

```
6504                    \cs_set:cpn { @@ _ \l_@@_letter_str } ##1
6505                      { \@@_v_custom_line:n { #1 } }
6506                  }
6507              }
6508          }
6509        \str_if_empty:NF \l_@@_command_str { \@@_h_custom_line:n { #1 } }
6510        \str_if_empty:NF \l_@@_ccommand_str { \@@_c_custom_line:n { #1 } }
6511      }

6512  \tl_const:Nn \c_@@_forbidden_letters_tl { lcrpmbVX|()[]!@<> }
6513  \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.

```
6514 \keys_define:nn { NiceMatrix / custom-line-bis }
6515   {
6516     multiplicity .int_set:N = \l_@@_multiplicity_int ,
6517     multiplicity .initial:n = 1 ,
6518     multiplicity .value_required:n = true ,
6519     color .code:n = \bool_set_true:N \l_@@_color_bool ,
6520     color .value_required:n = true ,
6521     tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6522     tikz .value_required:n = true ,
6523     dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6524     dotted .value_forbidden:n = true ,
6525     total-width .code:n = { } ,
6526     total-width .value_required:n = true ,
6527     width .code:n = { } ,
6528     width .value_required:n = true ,
6529     sep-color .code:n = { } ,
6530     sep-color .value_required:n = true ,
6531     unknown .code:n = \@@_error:n { Unknown~key~for~custom-line }
6532   }
```

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

```
6533 \bool_new:N \l_@@_dotted_rule_bool
6534 \bool_new:N \l_@@_tikz_rule_bool
6535 \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`.

```
6536 \keys_define:nn { NiceMatrix / custom-line-width }
6537   {
6538     multiplicity .int_set:N = \l_@@_multiplicity_int ,
6539     multiplicity .initial:n = 1 ,
6540     multiplicity .value_required:n = true ,
6541     tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
6542     total-width .code:n = \dim_set:Nn \l_@@_rule_width_dim { #1 }
6543                           \bool_set_true:N \l_@@_total_width_bool ,
6544     total-width .value_required:n = true ,
6545     width .meta:n = { total-width = #1 } ,
6546     dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
6547   }
```

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`).

```
6548 \cs_new_protected:Npn \@@_h_custom_line:n #1
6549   {
```

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`).

```
6550     \cs_set:cpn { nicematrix - \l_@@_command_str } { \Hline [ #1 ] }
6551     \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_command_str
6552   }
```

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`).

153

```
6553 \cs_new_protected:Npn \@@_c_custom_line:n #1
6554   {
```
Here, we need an expandable command since it begins with an \noalign.
```
6555     \exp_args:Nc \NewExpandableDocumentCommand
6556       { nicematrix - \l_@@_ccommand_str }
6557       { O { } m }
6558       {
6559         \noalign
6560           {
6561             \@@_compute_rule_width:n { #1 , ##1 }
6562             \skip_vertical:n { \l_@@_rule_width_dim }
6563             \clist_map_inline:nn
6564               { ##2 }
6565               { \@@_c_custom_line_i:nn { #1 , ##1 } { ####1 } }
6566           }
6567       }
6568     \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_ccommand_str
6569   }
```
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*.
```
6570 \cs_new_protected:Npn \@@_c_custom_line_i:nn #1 #2
6571   {
6572     \str_if_in:nnTF { #2 } { - }
6573       { \@@_cut_on_hyphen:w #2 \q_stop }
6574       { \@@_cut_on_hyphen:w #2 - #2 \q_stop }
6575     \tl_gput_right:Nx \g_@@_pre_code_after_tl
6576       {
6577         \@@_hline:n
6578           {
6579             #1 ,
6580             start = \l_tmpa_tl ,
6581             end = \l_tmpb_tl ,
6582             position = \int_eval:n { \c@iRow + 1 } ,
6583             total-width = \dim_use:N \l_@@_rule_width_dim
6584           }
6585       }
6586   }
6587 \cs_new_protected:Npn \@@_compute_rule_width:n #1
6588   {
6589     \bool_set_false:N \l_@@_tikz_rule_bool
6590     \bool_set_false:N \l_@@_total_width_bool
6591     \bool_set_false:N \l_@@_dotted_rule_bool
6592     \keys_set_known:nn { NiceMatrix / custom-line-width } { #1 }
6593     \bool_if:NF \l_@@_total_width_bool
6594       {
6595         \bool_if:NTF \l_@@_dotted_rule_bool
6596           { \dim_set:Nn \l_@@_rule_width_dim { 2 \l_@@_xdots_radius_dim } }
6597           {
6598             \bool_if:NF \l_@@_tikz_rule_bool
6599               {
6600                 \dim_set:Nn \l_@@_rule_width_dim
6601                   {
6602                     \arrayrulewidth * \l_@@_multiplicity_int
6603                     + \doublerulesep * ( \l_@@_multiplicity_int - 1 )
6604                   }
6605               }
6606           }
6607       }
6608   }
6609 \cs_new_protected:Npn \@@_v_custom_line:n #1
6610   {
6611     \@@_compute_rule_width:n { #1 }
```

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

```
6612        \tl_gput_right:Nx \g_@@_array_preamble_tl
6613          { \exp_not:N ! { \skip_horizontal:n { \dim_use:N \l_@@_rule_width_dim } } }
6614        \tl_gput_right:Nx \g_@@_pre_code_after_tl
6615          {
6616            \@@_vline:n
6617              {
6618                #1 ,
6619                position = \int_eval:n { \c@jCol + 1 } ,
6620                total-width = \dim_use:N \l_@@_rule_width_dim
6621              }
6622          }
6623        \@@_rec_preamble:n
6624      }
6625    \@@_custom_line:n
6626      { 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`.

```
6627    \cs_new_protected:Npn \@@_test_hline_in_block:nnnnn #1 #2 #3 #4 #5
6628      {
6629        \int_compare:nNnT \l_tmpa_tl > { #1 }
6630          {
6631            \int_compare:nNnT \l_tmpa_tl < { #3 + 1 }
6632              {
6633                \int_compare:nNnT \l_tmpb_tl > { #2 - 1 }
6634                  {
6635                    \int_compare:nNnT \l_tmpb_tl < { #4 + 1 }
6636                      { \bool_gset_false:N \g_tmpa_bool }
6637                  }
6638              }
6639          }
6640      }
```

The same for vertical rules.

```
6641    \cs_new_protected:Npn \@@_test_vline_in_block:nnnnn #1 #2 #3 #4 #5
6642      {
6643        \int_compare:nNnT \l_tmpa_tl > { #1 - 1 }
6644          {
6645            \int_compare:nNnT \l_tmpa_tl < { #3 + 1 }
6646              {
6647                \int_compare:nNnT \l_tmpb_tl > { #2 }
6648                  {
6649                    \int_compare:nNnT \l_tmpb_tl < { #4 + 1 }
6650                      { \bool_gset_false:N \g_tmpa_bool }
6651                  }
6652              }
6653          }
6654      }
6655    \cs_new_protected:Npn \@@_test_hline_in_stroken_block:nnnn #1 #2 #3 #4
6656      {
6657        \int_compare:nNnT \l_tmpb_tl > { #2 - 1 }
6658          {
6659            \int_compare:nNnT \l_tmpb_tl < { #4 + 1 }
6660              {
6661                \int_compare:nNnTF \l_tmpa_tl = { #1 }
6662                  { \bool_gset_false:N \g_tmpa_bool }
6663                  {
```

```
6664            \int_compare:nNnT \l_tmpa_tl = { #3 + 1 }
6665              { \bool_gset_false:N \g_tmpa_bool }
6666          }
6667        }
6668      }
6669    }
6670  \cs_new_protected:Npn \@@_test_vline_in_stroken_block:nnnn #1 #2 #3 #4
6671    {
6672      \int_compare:nNnT \l_tmpa_tl > { #1 - 1 }
6673        {
6674          \int_compare:nNnT \l_tmpa_tl < { #3 + 1 }
6675            {
6676              \int_compare:nNnTF \l_tmpb_tl = { #2 }
6677                { \bool_gset_false:N \g_tmpa_bool }
6678                {
6679                  \int_compare:nNnT \l_tmpb_tl = { #4 + 1 }
6680                    { \bool_gset_false:N \g_tmpa_bool }
6681                }
6682            }
6683        }
6684    }
```

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.

```
6685  \cs_new_protected:Npn \@@_compute_corners:
6686    {
```

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.

```
6687      \seq_clear_new:N \l_@@_corners_cells_seq
6688      \clist_map_inline:Nn \l_@@_corners_clist
6689        {
6690          \str_case:nnF { ##1 }
6691            {
6692              { NW }
6693              { \@@_compute_a_corner:nnnnnn 1 1 1 1 \c@iRow \c@jCol }
6694              { NE }
6695              { \@@_compute_a_corner:nnnnnn 1 \c@jCol 1 { -1 } \c@iRow 1 }
6696              { SW }
6697              { \@@_compute_a_corner:nnnnnn \c@iRow 1 { -1 } 1 1 \c@jCol }
6698              { SE }
6699              { \@@_compute_a_corner:nnnnnn \c@iRow \c@jCol { -1 } { -1 } 1 1 }
6700            }
6701            { \@@_error:nn { bad~corner } { ##1 } } }
6702        }
```

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

```
6703      \seq_if_empty:NF \l_@@_corners_cells_seq
6704        {
```

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.

```
6705          \tl_gput_right:Nx \g_@@_aux_tl
6706            {
6707              \seq_set_from_clist:Nn \exp_not:N \l_@@_corners_cells_seq
```

```
6708          { \seq_use:Nnnn \l_@@_corners_cells_seq , , , }
6709        }
6710      }
6711    }
```

"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.

```
6712 \cs_new_protected:Npn \@@_compute_a_corner:nnnnnn #1 #2 #3 #4 #5 #6
6713   {
```

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.

```
6714      \bool_set_false:N \l_tmpa_bool
6715      \int_zero_new:N \l_@@_last_empty_row_int
6716      \int_set:Nn \l_@@_last_empty_row_int { #1 }
6717      \int_step_inline:nnnn { #1 } { #3 } { #5 }
6718        {
6719          \@@_test_if_cell_in_a_block:nn { ##1 } { \int_eval:n { #2 } }
6720          \bool_lazy_or:nnTF
6721            {
6722              \cs_if_exist_p:c
6723                { pgf @ sh @ ns @ \@@_env: - ##1 - \int_eval:n { #2 } }
6724            }
6725            \l_tmpb_bool
6726            { \bool_set_true:N \l_tmpa_bool }
6727            {
6728              \bool_if:NF \l_tmpa_bool
6729                { \int_set:Nn \l_@@_last_empty_row_int { ##1 } }
6730            }
6731        }
```

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

```
6732      \bool_set_false:N \l_tmpa_bool
6733      \int_zero_new:N \l_@@_last_empty_column_int
6734      \int_set:Nn \l_@@_last_empty_column_int { #2 }
6735      \int_step_inline:nnnn { #2 } { #4 } { #6 }
6736        {
6737          \@@_test_if_cell_in_a_block:nn { \int_eval:n { #1 } } { ##1 }
6738          \bool_lazy_or:nnTF
6739            \l_tmpb_bool
6740            {
6741              \cs_if_exist_p:c
6742                { pgf @ sh @ ns @ \@@_env: - \int_eval:n { #1 } - ##1 }
6743            }
6744            { \bool_set_true:N \l_tmpa_bool }
6745            {
6746              \bool_if:NF \l_tmpa_bool
6747                { \int_set:Nn \l_@@_last_empty_column_int { ##1 } }
6748            }
6749        }
```

Now, we loop over the rows.

```
6750        \int_step_inline:nnnn { #1 } { #3 } \l_@@_last_empty_row_int
6751          {
```

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

```
6752            \bool_set_false:N \l_tmpa_bool
6753            \int_step_inline:nnnn { #2 } { #4 } \l_@@_last_empty_column_int
6754              {
6755                \@@_test_if_cell_in_a_block:nn { ##1 } { ####1 }
6756                \bool_lazy_or:nnTF
6757                  \l_tmpb_bool
6758                  {
6759                    \cs_if_exist_p:c
6760                      { pgf @ sh @ ns @ \@@_env: - ##1 - ####1 }
6761                  }
6762                  { \bool_set_true:N \l_tmpa_bool }
6763                  {
6764                    \bool_if:NF \l_tmpa_bool
6765                      {
6766                        \int_set:Nn \l_@@_last_empty_column_int { ####1 }
6767                        \seq_put_right:Nn
6768                          \l_@@_corners_cells_seq
6769                          { ##1 - ####1 }
6770                      }
6771                  }
6772              }
6773          }
6774      }
```

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`).

```
6775 \cs_new_protected:Npn \@@_test_if_cell_in_a_block:nn #1 #2
6776   {
6777     \int_set:Nn \l_tmpa_int { #1 }
6778     \int_set:Nn \l_tmpb_int { #2 }
6779     \bool_set_false:N \l_tmpb_bool
6780     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
6781       { \@@_test_if_cell_in_block:nnnnnnn \l_tmpa_int \l_tmpb_int ##1 }
6782   }
6783 \cs_set_protected:Npn \@@_test_if_cell_in_block:nnnnnnn #1 #2 #3 #4 #5 #6 #7
6784   {
6785     \int_compare:nNnF { #3 } > { #1 }
6786       {
6787         \int_compare:nNnF { #1 } > { #5 }
6788           {
6789             \int_compare:nNnF { #4 } > { #2 }
6790               {
6791                 \int_compare:nNnF { #2 } > { #6 }
6792                   { \bool_set_true:N \l_tmpb_bool }
6793               }
6794           }
6795       }
6796   }
```

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.

```
6797 \bool_new:N \l_@@_block_auto_columns_width_bool
```

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

```
6798 \keys_define:nn { NiceMatrix / NiceMatrixBlock }
6799   {
6800     auto-columns-width .code:n =
6801       {
6802         \bool_set_true:N \l_@@_block_auto_columns_width_bool
6803         \dim_gzero_new:N \g_@@_max_cell_width_dim
6804         \bool_set_true:N \l_@@_auto_columns_width_bool
6805       }
6806   }
```


```
6807 \NewDocumentEnvironment { NiceMatrixBlock } { ! O { } }
6808   {
6809     \int_gincr:N \g_@@_NiceMatrixBlock_int
6810     \dim_zero:N \l_@@_columns_width_dim
6811     \keys_set:nn { NiceMatrix / NiceMatrixBlock } { #1 }
6812     \bool_if:NT \l_@@_block_auto_columns_width_bool
6813       {
6814         \cs_if_exist:cT
6815           { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
6816           {
6817             % is \exp_args:NNe mandatory?
6818             \exp_args:NNe \dim_set:Nn \l_@@_columns_width_dim
6819               {
6820                 \use:c
6821                   { @@_max_cell_width _ \int_use:N \g_@@_NiceMatrixBlock_int }
6822               }
6823           }
6824       }
6825   }
```

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).

```
6826   {
6827     \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.

```
6828     { \int_gdecr:N \g_@@_NiceMatrixBlock_int }
6829     {
6830       \bool_if:NT \l_@@_block_auto_columns_width_bool
6831         {
6832           \iow_shipout:Nn \@mainaux \ExplSyntaxOn
6833           \iow_shipout:Nx \@mainaux
6834             {
6835               \cs_gset:cpn
6836                 { @@ _ 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.

```
6837                 { \dim_eval:n { \g_@@_max_cell_width_dim + \arrayrulewidth } }
6838             }
6839           \iow_shipout:Nn \@mainaux \ExplSyntaxOff
6840         }
6841     }
6842     \ignorespacesafterend
6843   }
```

26 The extra nodes

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

```
6844 \cs_generate_variant:Nn \dim_min:nn { v n }
6845 \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).

```
6846 \cs_new_protected:Npn \@@_create_extra_nodes:
6847   {
6848     \bool_if:nTF \l_@@_medium_nodes_bool
6849       {
6850         \bool_if:NTF \l_@@_large_nodes_bool
6851           \@@_create_medium_and_large_nodes:
6852           \@@_create_medium_nodes:
6853       }
6854       { \bool_if:NT \l_@@_large_nodes_bool \@@_create_large_nodes: }
6855   }
```

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`.

```
6856 \cs_new_protected:Npn \@@_computations_for_medium_nodes:
6857   {
6858     \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6859       {
6860         \dim_zero_new:c { l_@@_row_\@@_i: _min_dim }
6861         \dim_set_eq:cN { l_@@_row_\@@_i: _min_dim } \c_max_dim
6862         \dim_zero_new:c { l_@@_row_\@@_i: _max_dim }
6863         \dim_set:cn { l_@@_row_\@@_i: _max_dim } { - \c_max_dim }
6864       }
6865     \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
6866       {
6867         \dim_zero_new:c { l_@@_column_\@@_j: _min_dim }
6868         \dim_set_eq:cN { l_@@_column_\@@_j: _min_dim } \c_max_dim
6869         \dim_zero_new:c { l_@@_column_\@@_j: _max_dim }
6870         \dim_set:cn { l_@@_column_\@@_j: _max_dim } { - \c_max_dim }
6871       }
```

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

```
6872     \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6873       {
6874         \int_step_variable:nnNn
6875           \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.

```
6876                 {
6877                   \cs_if_exist:cT
6878                     { 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.

```
6879                       {
6880                         \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { south~west }
6881                         \dim_set:cn { l_@@_row_\@@_i: _min_dim}
6882                           { \dim_min:vn { l_@@_row _ \@@_i: _min_dim } \pgf@y }
6883                         \seq_if_in:NxF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
6884                           {
6885                             \dim_set:cn { l_@@_column _ \@@_j: _min_dim}
6886                               { \dim_min:vn { l_@@_column _ \@@_j: _min_dim } \pgf@x }
6887                           }
```

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.

```
6888                         \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { north~east }
6889                         \dim_set:cn { l_@@_row _ \@@_i: _ max_dim }
6890                           { \dim_max:vn { l_@@_row _ \@@_i: _ max_dim } \pgf@y }
6891                         \seq_if_in:NxF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
6892                           {
6893                             \dim_set:cn { l_@@_column _ \@@_j: _ max_dim }
6894                               { \dim_max:vn { l_@@_column _ \@@_j: _max_dim } \pgf@x }
6895                           }
6896                       }
6897                   }
6898                 }
```

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

```
6899         \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6900           {
6901             \dim_compare:nNnT
6902               { \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } } = \c_max_dim
6903               {
6904                 \@@_qpoint:n { row - \@@_i: - base }
6905                 \dim_set:cn { l_@@_row _ \@@_i: _ max _ dim } \pgf@y
6906                 \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim } \pgf@y
6907               }
6908           }
6909         \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
6910           {
6911             \dim_compare:nNnT
6912               { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } } = \c_max_dim
6913               {
6914                 \@@_qpoint:n { col - \@@_j: }
6915                 \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim } \pgf@y
6916                 \dim_set:cn { l_@@_column _ \@@_j: _ min _ dim } \pgf@y
6917               }
6918           }
6919       }
```

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

```
6920   \cs_new_protected:Npn \@@_create_medium_nodes:
6921     {
6922       \pgfpicture
6923         \pgfrememberpicturepositiononpagetrue
6924         \pgf@relevantforpicturesizefalse
6925         \@@_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".

```
6926        \cs_set_nopar:Npn \l_@@_suffix_tl { -medium }
6927        \@@_create_nodes:
6928        \endpgfpicture
6929    }
```

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:`.

```
6930 \cs_new_protected:Npn \@@_create_large_nodes:
6931    {
6932      \pgfpicture
6933        \pgfremberpicturepositiononpagetrue
6934        \pgf@relevantforpicturesizefalse
6935        \@@_computations_for_medium_nodes:
6936        \@@_computations_for_large_nodes:
6937        \cs_set_nopar:Npn \l_@@_suffix_tl { - large }
6938        \@@_create_nodes:
6939      \endpgfpicture
6940    }
6941 \cs_new_protected:Npn \@@_create_medium_and_large_nodes:
6942    {
6943      \pgfpicture
6944        \pgfremberpicturepositiononpagetrue
6945        \pgf@relevantforpicturesizefalse
6946        \@@_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".

```
6947        \cs_set_nopar:Npn \l_@@_suffix_tl { - medium }
6948        \@@_create_nodes:
6949        \@@_computations_for_large_nodes:
6950        \cs_set_nopar:Npn \l_@@_suffix_tl { - large }
6951        \@@_create_nodes:
6952      \endpgfpicture
6953    }
```

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.

```
6954 \cs_new_protected:Npn \@@_computations_for_large_nodes:
6955    {
6956      \int_set_eq:NN \l_@@_first_row_int \c_one_int
6957      \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`.

```
6958        \int_step_variable:nNn { \c@iRow - 1 } \@@_i:
6959          {
6960            \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim }
6961              {
6962                (
6963                  \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } +
6964                  \dim_use:c { l_@@_row _ \int_eval:n { \@@_i: + 1 }  _ max _ dim }
6965                )
6966                / 2
6967              }
```

---

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

```
6968        \dim_set_eq:cc { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
6969            { l_@@_row_\@@_i: _min_dim }
6970        }
6971    \int_step_variable:nNn { \c@jCol - 1 } \@@_j:
6972        {
6973        \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim }
6974            {
6975            (
6976                \dim_use:c { l_@@_column _ \@@_j: _ max _ dim } +
6977                \dim_use:c
6978                    { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
6979            )
6980            / 2
6981            }
6982        \dim_set_eq:cc { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
6983            { l_@@_column _ \@@_j: _ max _ dim }
6984        }
```

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

```
6985        \dim_sub:cn
6986            { l_@@_column _ 1 _ min _ dim }
6987            \l_@@_left_margin_dim
6988        \dim_add:cn
6989            { l_@@_column _ \int_use:N \c@jCol _ max _ dim }
6990            \l_@@_right_margin_dim
6991    }
```

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`).

```
6992 \cs_new_protected:Npn \@@_create_nodes:
6993    {
6994    \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
6995        {
6996        \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
6997            {
```

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

```
6998            \@@_pgf_rect_node:nnnnn
6999                { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7000                { \dim_use:c { l_@@_column_ \@@_j: _min_dim } }
7001                { \dim_use:c { l_@@_row_ \@@_i: _min_dim } }
7002                { \dim_use:c { l_@@_column_ \@@_j: _max_dim } }
7003                { \dim_use:c { l_@@_row_ \@@_i: _max_dim } }
7004            \str_if_empty:NF \l_@@_name_str
7005                {
7006                \pgfnodealias
7007                    { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
7008                    { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7009                }
7010            }
7011        }
```

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.

```
7012        \seq_map_pairwise_function:NNN
7013        \g_@@_multicolumn_cells_seq
7014        \g_@@_multicolumn_sizes_seq
7015        \@@_node_for_multicolumn:nn
7016    }
```

```
7017  \cs_new_protected:Npn \@@_extract_coords_values: #1 - #2 \q_stop
7018    {
7019      \cs_set_nopar:Npn \@@_i: { #1 }
7020      \cs_set_nopar:Npn \@@_j: { #2 }
7021    }
```

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 $i\text{-}j$ and the second is the value of n (the length of the "multi-cell").

```
7022  \cs_new_protected:Npn \@@_node_for_multicolumn:nn #1 #2
7023    {
7024      \@@_extract_coords_values: #1 \q_stop
7025      \@@_pgf_rect_node:nnnnn
7026        { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
7027        { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } }
7028        { \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } }
7029        { \dim_use:c { l_@@_column _ \int_eval:n { \@@_j: +#2-1 } _ max _ dim } }
7030        { \dim_use:c { l_@@_row _ \@@_i: _ max _ dim } }
7031      \str_if_empty:NF \l_@@_name_str
7032        {
7033          \pgfnodealias
7034            { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
7035            { \int_use:N \g_@@_env_int - \@@_i: - \@@_j: \l_@@_suffix_tl}
7036        }
7037    }
```

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).

```
7038  \keys_define:nn { NiceMatrix / Block / FirstPass }
7039    {
7040      j .code:n = \str_set:Nn \l_@@_hpos_block_str j
7041                  \bool_set_true:N \l_@@_p_block_bool ,
7042      j .value_forbidden:n = true ,
7043      l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7044      l .value_forbidden:n = true ,
7045      r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7046      r .value_forbidden:n = true ,
7047      c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7048      c .value_forbidden:n = true ,
7049      L .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7050      L .value_forbidden:n = true ,
7051      R .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7052      R .value_forbidden:n = true ,
7053      C .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7054      C .value_forbidden:n = true ,
7055      t .code:n = \str_set:Nn \l_@@_vpos_block_str t ,
7056      t .value_forbidden:n = true ,
7057      T .code:n = \str_set:Nn \l_@@_vpos_block_str T ,
7058      T .value_forbidden:n = true ,
7059      b .code:n = \str_set:Nn \l_@@_vpos_block_str b ,
7060      b .value_forbidden:n = true ,
7061      B .code:n = \str_set:Nn \l_@@_vpos_block_str B ,
7062      B .value_forbidden:n = true ,
```

```
7063        p .code:n = \bool_set_true:N \l_@@_p_block_bool ,
7064        p .value_forbidden:n = true ,
7065        color .code:n =
7066          \@@_color:n { #1 }
7067          \tl_set_rescan:Nnn
7068            \l_@@_draw_tl
7069            { \char_set_catcode_other:N ! }
7070            { #1 } ,
7071        color .value_required:n = true ,
7072        respect-arraystretch .code:n =
7073          \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
7074        respect-arraystretch .value_forbidden:n = true ,
7075      }
```

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.

```
7076  \cs_new_protected:Npn \@@_Block: { \@@_collect_options:n { \@@_Block_i: } }
```

```
7077  \NewExpandableDocumentCommand \@@_Block_i: { m m D < > { } +m }
7078    {
```

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).

```
7079      \peek_remove_spaces:n
7080        {
7081          \tl_if_blank:nTF { #2 }
7082            { \@@_Block_ii:nnnnn \c_one_int \c_one_int }
7083            {
7084              \int_compare:nNnTF { \char_value_catcode:n { 45 } } = { 13 }
7085              \@@_Block_i_czech \@@_Block_i
7086              #2 \q_stop
7087            }
7088          { #1 } { #3 } { #4 }
7089        }
7090    }
```

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

```
7091  \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`.

```
7092  {
7093    \char_set_catcode_active:N -
7094    \cs_new:Npn \@@_Block_i_czech #1-#2 \q_stop { \@@_Block_ii:nnnnn { #1 } { #2 } }
7095  }
```

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.

```
7096  \cs_new_protected:Npn \@@_Block_ii:nnnnn #1 #2 #3 #4 #5
7097    {
```

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).

```
7098      \bool_lazy_or:nnTF
7099        { \tl_if_blank_p:n { #1 } }
```

```
7100        { \str_if_eq_p:nn { #1 } { * } }
7101        { \int_set:Nn \l_tmpa_int { 100 } }
7102        { \int_set:Nn \l_tmpa_int { #1 } }
7103     \bool_lazy_or:nnTF
7104        { \tl_if_blank_p:n { #2 } }
7105        { \str_if_eq_p:nn { #2 } { * } }
7106        { \int_set:Nn \l_tmpb_int { 100 } }
7107        { \int_set:Nn \l_tmpb_int { #2 } }
```

If the block is mono-column.

```
7108        \int_compare:nNnTF \l_tmpb_int = \c_one_int
7109          {
7110            \tl_if_empty:NTF \l_@@_hpos_cell_tl
7111              { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_c_str }
7112              { \str_set:NV \l_@@_hpos_block_str \l_@@_hpos_cell_tl }
7113          }
7114          { \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.

```
7115        \keys_set_known:nn { NiceMatrix / Block / FirstPass } { #3 }
7116        \tl_set:Nx \l_tmpa_tl
7117          {
7118            { \int_use:N \c@iRow }
7119            { \int_use:N \c@jCol }
7120            { \int_eval:n { \c@iRow + \l_tmpa_int - 1 } }
7121            { \int_eval:n { \c@jCol + \l_tmpb_int - 1 } }
7122          }
```

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).

```
7123        \bool_set_false:N \l_tmpa_bool
7124        \bool_if:NT \l_@@_amp_in_blocks_bool
7125          { \tl_if_in:nnT { #5 } { & } { \bool_set_true:N \l_tmpa_bool } }
7126        \bool_if:NTF \l_tmpa_bool
7127          { \exp_args:Nee \@@_Block_vii:nnnnn }
7128          {
7129            \bool_if:NTF \l_@@_p_block_bool
7130              { \exp_args:Nee \@@_Block_vi:nnnnn }
7131              {
7132                \bool_if:nTF
7133                  {
7134                    (
7135                      \int_compare_p:nNn \l_tmpa_int = \c_one_int
7136                        ||
7137                      \int_compare_p:nNn \l_tmpb_int = \c_one_int
7138                    )
7139                    && ! \tl_if_empty_p:n { #5 }
```

For the blocks mono-column, we will compose right now in a box in order to compute its width and take that width into account for the width of the column. However, if the column is a `X` column, we should not do that since the width is determined by another way. This should be the same for the `p`, `m` and `b` columns and we should modify that point. However, for the `X` column, it's imperative. Otherwise, the process for the determination of the widths of the columns will be wrong.

```
7140                    && ! \l_@@_X_bool
7141                  }
7142                  { \exp_args:Nee \@@_Block_iv:nnnnn }
7143                  { \exp_args:Nee \@@_Block_v:nnnnn }
```

```
7144            }
7145          }
7146        { \l_tmpa_int } { \l_tmpb_int } { #3 } { #4 } { #5 }
7147      }
```

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.

```
7148    \cs_new_protected:Npn \@@_Block_iv:nnnnn #1 #2 #3 #4 #5
7149      {
7150        \int_gincr:N \g_@@_block_box_int
7151        \cs_set_protected_nopar:Npn \diagbox ##1 ##2
7152          {
7153            \tl_gput_right:Nx \g_@@_pre_code_after_tl
7154              {
7155                \@@_actually_diagbox:nnnnnn
7156                  { \int_use:N \c@iRow }
7157                  { \int_use:N \c@jCol }
7158                  { \int_eval:n { \c@iRow + #1 - 1 } }
7159                  { \int_eval:n { \c@jCol + #2 - 1 } }
7160                  { \g_@@_row_style_tl \exp_not:n { ##1 } }
7161                  { \g_@@_row_style_tl \exp_not:n { ##2 } }
7162              }
7163          }
7164        \box_gclear_new:c
7165          { 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!

```
7166        \hbox_gset:cn
7167          { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7168          {
```

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}).

```
7169            \tl_if_empty:NTF \l_@@_color_tl
7170              { \int_compare:nNnT { #2 } = \c_one_int \set@color }
7171              { \@@_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.

```
7172            \int_compare:nNnT { #1 } = \c_one_int
7173              {
7174                \int_if_zero:nTF \c@iRow
7175                  \l_@@_code_for_first_row_tl
7176                  {
7177                    \int_compare:nNnT \c@iRow = \l_@@_last_row_int
7178                      \l_@@_code_for_last_row_tl
7179                  }
7180                \g_@@_row_style_tl
7181              }
```

The following command will be no-op when `respect-arraystretch` is in force.

```
7182        \@@_reset_arraystretch:
7183        \dim_zero:N \extrarowheight
```

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

```
7184        #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.).

```
7185        \@@_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}`.

```
7186        \bool_if:NTF \l_@@_tabular_bool
7187          {
7188            \bool_lazy_all:nTF
7189              {
7190                { \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.

```
7191                { ! \dim_compare_p:nNn \l_@@_col_width_dim < \c_zero_dim }
7192                { ! \g_@@_rotate_bool }
7193              }
```

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

```
7194              {
7195                \use:e
7196                  {
7197                    \exp_not:N \begin { minipage }%
7198                      [ \str_lowercase:o \l_@@_vpos_block_str ]
7199                      { \l_@@_col_width_dim }
7200                    \str_case:on \l_@@_hpos_block_str
7201                      { c \centering r \raggedleft l \raggedright }
7202                  }
7203                #5
7204                \end { minipage }
7205              }
```

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

```
7206              {
7207                \use:e
7208                  {
7209                    \exp_not:N \begin { tabular }%
7210                      [ \str_lowercase:o \l_@@_vpos_block_str ]
7211                      { @ { } \l_@@_hpos_block_str @ { } }
7212                  }
7213                #5
7214                \end { tabular }
7215              }
7216          }
```

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

```
7217          {
7218            \c_math_toggle_token
7219            \use:e
7220              {
7221                \exp_not:N \begin { array }%
7222                  [ \str_lowercase:o \l_@@_vpos_block_str ]
7223                  { @ { } \l_@@_hpos_block_str @ { } }
7224              }
```

```
7225                    #5
7226                \end { array }
7227                \c_math_toggle_token
7228            }
7229        }
```

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).

```
7230        \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.

```
7231        \int_compare:nNnT { #2 } = \c_one_int
7232            {
7233            \dim_gset:Nn \g_@@_blocks_wd_dim
7234                {
7235                \dim_max:nn
7236                    \g_@@_blocks_wd_dim
7237                    {
7238                    \box_wd:c
7239                        { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7240                    }
7241                }
7242            }
```

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.

```
7243        \int_compare:nNnT { #1 } = \c_one_int
7244            {
7245            \dim_gset:Nn \g_@@_blocks_ht_dim
7246                {
7247                \dim_max:nn
7248                    \g_@@_blocks_ht_dim
7249                    {
7250                    \box_ht:c
7251                        { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7252                    }
7253                }
7254            \dim_gset:Nn \g_@@_blocks_dp_dim
7255                {
7256                \dim_max:nn
7257                    \g_@@_blocks_dp_dim
7258                    {
7259                    \box_dp:c
7260                        { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7261                    }
7262                }
7263            }
7264        \seq_gput_right:Nx \g_@@_blocks_seq
7265            {
7266                \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.

```
7267            {
7268            \exp_not:n { #3 } ,
7269            \l_@@_hpos_block_str ,
```

Now, we put a key for the vertical alignment.

```
7270            \bool_if:NT \g_@@_rotate_bool
7271                {
```

```
7272              \bool_if:NTF \g_@@_rotate_c_bool
7273                { m }
7274                { \int_compare:nNnT \c@iRow = \l_@@_last_row_int T }
7275            }

7276

7277          }
7278          {
7279            \box_use_drop:c
7280              { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7281          }
7282        }
7283      \bool_set_false:N \g_@@_rotate_c_bool
7284    }


7285 \cs_new:Npn \@@_adjust_hpos_rotate:
7286    {
7287      \bool_if:NT \g_@@_rotate_bool
7288        {
7289          \str_set:Nx \l_@@_hpos_block_str
7290            {
7291              \bool_if:NTF \g_@@_rotate_c_bool
7292                { c }
7293                {
7294                  \str_case:onF \l_@@_vpos_block_str
7295                    { b l B l t r T r }
7296                    { \int_compare:nNnTF \c@iRow = \l_@@_last_row_int r l }
7297                }
7298            }
7299        }
7300    }
```

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

```
7301 \cs_new_protected:Npn \@@_rotate_box_of_block:
7302    {
7303      \box_grotate:cn
7304        { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7305        { 90 }
7306      \int_compare:nNnT \c@iRow = \l_@@_last_row_int
7307        {
7308          \vbox_gset_top:cn
7309            { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7310            {
7311              \skip_vertical:n { 0.8 ex }
7312              \box_use:c
7313                { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7314            }
7315        }
7316      \bool_if:NT \g_@@_rotate_c_bool
7317        {
7318          \hbox_gset:cn
7319            { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7320            {
7321              \c_math_toggle_token
7322              \vcenter
7323                {
7324                  \box_use:c
7325                  { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
7326                }
7327              \c_math_toggle_token
7328            }
7329        }
7330    }
```

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:nnnnnn).

#1 is i (the number of rows of the block), #2 is j (the number of columns of the block), #3 is the list of *key=values* pairs, #4 are the tokens to put before the math mode and before the composition of the block and #5 is the label (=content) of the block.

```
7331 \cs_new_protected:Npn \@@_Block_v:nnnnn #1 #2 #3 #4 #5
7332   {
7333     \seq_gput_right:Nx \g_@@_blocks_seq
7334       {
7335         \l_tmpa_tl
7336         { \exp_not:n { #3 } }
7337         {
7338           \bool_if:NTF \l_@@_tabular_bool
7339             {
7340               \group_begin:
```

The following command will be no-op when respect-arraystretch is in force.

```
7341               \@@_reset_arraystretch:
7342               \exp_not:n
7343                 {
7344                   \dim_zero:N \extrarowheight
7345                   #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).

```
7346                   \bool_if:NT \c_@@_testphase_table_bool
7347                     { \tag_stop:n { table } }
7348                   \use:e
7349                     {
7350                       \exp_not:N \begin { tabular } [ \l_@@_vpos_block_str ]
7351                       { @ { } \l_@@_hpos_block_str @ { } }
7352                     }
7353                   #5
7354                   \end { tabular }
7355                 }
7356               \group_end:
7357             }
```

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

```
7358             {
7359               \group_begin:
```

The following will be no-op when respect-arraystretch is in force.

```
7360               \@@_reset_arraystretch:
7361               \exp_not:n
7362                 {
7363                   \dim_zero:N \extrarowheight
7364                   #4
7365                   \c_math_toggle_token
7366                   \use:e
7367                     {
7368                       \exp_not:N \begin { array } [ \l_@@_vpos_block_str ]
7369                       { @ { } \l_@@_hpos_block_str @ { } }
7370                     }
7371                   #5
7372                   \end { array }
7373                   \c_math_toggle_token
7374                 }
7375               \group_end:
```

171

```
7376                }
7377            }
7378        }
7379    }
```

The following macro is for the case of a `\Block` which uses the key p.

```
7380 \cs_new_protected:Npn \@@_Block_vi:nnnnn #1 #2 #3 #4 #5
7381    {
7382      \seq_gput_right:Nx \g_@@_blocks_seq
7383        {
7384          \l_tmpa_tl
7385          { \exp_not:n { #3 } }
7386          {
7387            \group_begin:
7388            \exp_not:n { #4 #5 }
7389            \group_end:
7390          }
7391        }
7392    }
```

The following macro is for the case of a `\Block` which uses the key p.

```
7393 \cs_new_protected:Npn \@@_Block_vii:nnnnn #1 #2 #3 #4 #5
7394    {
7395      \seq_gput_right:Nx \g_@@_blocks_seq
7396        {
7397          \l_tmpa_tl
7398          { \exp_not:n { #3 } }
7399          { \exp_not:n { #4 #5 } }
7400        }
7401    }
```

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).

```
7402 \keys_define:nn { NiceMatrix / Block / SecondPass }
7403    {
7404      tikz .code:n =
7405        \IfPackageLoadedTF { tikz }
7406          { \seq_put_right:Nn \l_@@_tikz_seq { { #1 } } }
7407          { \@@_error:n { tikz~key~without~tikz } } ,
7408      tikz .value_required:n = true ,
7409      fill .code:n =
7410        \tl_set_rescan:Nnn
7411          \l_@@_fill_tl
7412          { \char_set_catcode_other:N ! }
7413          { #1 } ,
7414      fill .value_required:n = true ,
7415      opacity .tl_set:N = \l_@@_opacity_tl ,
7416      opacity .value_required:n = true ,
7417      draw .code:n =
7418        \tl_set_rescan:Nnn
7419          \l_@@_draw_tl
7420          { \char_set_catcode_other:N ! }
7421          { #1 } ,
7422      draw .default:n = default ,
7423      rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
7424      rounded-corners .default:n = 4 pt ,
7425      color .code:n =
7426        \@@_color:n { #1 }
7427        \tl_set_rescan:Nnn
7428          \l_@@_draw_tl
```

```
7429          { \char_set_catcode_other:N ! }
7430          { #1 } ,
7431      borders .clist_set:N = \l_@@_borders_clist ,
7432      borders .value_required:n = true ,
7433      hvlines .meta:n = { vlines , hlines } ,
7434      vlines .bool_set:N = \l_@@_vlines_block_bool,
7435      vlines .default:n = true ,
7436      hlines .bool_set:N = \l_@@_hlines_block_bool,
7437      hlines .default:n = true ,
7438      line-width .dim_set:N = \l_@@_line_width_dim ,
7439      line-width .value_required:n = true ,
```

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

```
7440      j .code:n = \str_set:Nn \l_@@_hpos_block_str j
7441                  \bool_set_true:N \l_@@_p_block_bool ,
7442      l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
7443      r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
7444      c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
7445      L .code:n = \str_set:Nn \l_@@_hpos_block_str l
7446                  \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7447      R .code:n = \str_set:Nn \l_@@_hpos_block_str r
7448                  \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7449      C .code:n = \str_set:Nn \l_@@_hpos_block_str c
7450                  \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
7451      t .code:n = \str_set:Nn \l_@@_vpos_block_str t ,
7452      T .code:n = \str_set:Nn \l_@@_vpos_block_str T ,
7453      b .code:n = \str_set:Nn \l_@@_vpos_block_str b ,
7454      B .code:n = \str_set:Nn \l_@@_vpos_block_str B ,
7455      m .code:n = \str_set:Nn \l_@@_vpos_block_str c ,
7456      m .value_forbidden:n = true ,
7457      v-center .meta:n = m ,
7458      p .code:n = \bool_set_true:N \l_@@_p_block_bool ,
7459      p .value_forbidden:n = true ,
7460      name .tl_set:N = \l_@@_block_name_str ,
7461      name .value_required:n = true ,
7462      name .initial:n = ,
7463      respect-arraystretch .code:n =
7464        \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
7465      respect-arraystretch .value_forbidden:n = true ,
7466      transparent .bool_set:N = \l_@@_transparent_bool ,
7467      transparent .default:n = true ,
7468      transparent .initial:n = false ,
7469      unknown .code:n = \@@_error:n { Unknown~key~for~Block }
7470    }
```

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.

```
7471 \cs_new_protected:Npn \@@_draw_blocks:
7472   {
7473     \bool_if:NTF \c_@@_tagging_array_bool
7474       { \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign: }
7475       { \cs_set_eq:NN \ialign \@@_old_ialign: }
7476     \seq_map_inline:Nn \g_@@_blocks_seq { \@@_Block_iv:nnnnnn ##1 }
7477   }

7478 \cs_new_protected:Npn \@@_Block_iv:nnnnnn #1 #2 #3 #4 #5 #6
7479   {
```

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

```
7480     \int_zero_new:N \l_@@_last_row_int
7481     \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.

```
7482      \int_compare:nNnTF { #3 } > { 99 }
7483        { \int_set_eq:NN \l_@@_last_row_int \c@iRow }
7484        { \int_set:Nn \l_@@_last_row_int { #3 } }
7485      \int_compare:nNnTF { #4 } > { 99 }
7486        { \int_set_eq:NN \l_@@_last_col_int \c@jCol }
7487        { \int_set:Nn \l_@@_last_col_int { #4 } }
7488      \int_compare:nNnTF \l_@@_last_col_int > \g_@@_col_total_int
7489        {
7490          \bool_lazy_and:nnTF
7491            \l_@@_preamble_bool
7492            {
7493              \int_compare_p:n
7494                { \l_@@_last_col_int <= \g_@@_static_num_of_col_int }
7495            }
7496            {
7497              \msg_error:nnnn { nicematrix } { Block~too~large~2 } { #1 } { #2 }
7498              \@@_msg_redirect_name:nn { Block~too~large~2 } { none }
7499              \@@_msg_redirect_name:nn { columns~not~used } { none }
7500            }
7501            { \msg_error:nnnn { nicematrix } { Block~too~large~1 } { #1 } { #2 } }
7502        }
7503        {
7504          \int_compare:nNnTF \l_@@_last_row_int > \g_@@_row_total_int
7505            { \msg_error:nnnn { nicematrix } { Block~too~large~1 } { #1 } { #2 } }
7506            { \@@_Block_v:nnnnnn { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } }
7507        }
7508    }
```

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

```
7509 \cs_new_protected:Npn \@@_Block_v:nnnnnn #1 #2 #3 #4 #5 #6
7510    {
```

The group is for the keys.

```
7511      \group_begin:
7512      \int_compare:nNnT { #1 } = { #3 }
7513        { \str_set:Nn \l_@@_vpos_block_str { t } }
7514      \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).

```
7515      \tl_if_in:nnT { #6 } { & } { \bool_set_true:N \l_@@_ampersand_bool }
7516      \bool_lazy_and:nnT
7517        \l_@@_vlines_block_bool
7518        { ! \l_@@_ampersand_bool }
7519        {
7520          \tl_gput_right:Nx \g_nicematrix_code_after_tl
7521            {
7522              \@@_vlines_block:nnn
7523                { \exp_not:n { #5 } }
7524                { #1 - #2 }
7525                { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7526            }
7527        }
7528      \bool_if:NT \l_@@_hlines_block_bool
7529        {
```

174

```
7530        \tl_gput_right:Nx \g_nicematrix_code_after_tl
7531          {
7532            \@@_hlines_block:nnn
7533              { \exp_not:n { #5 } }
7534              { #1 - #2 }
7535              { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7536          }
7537        }
7538      \bool_if:NF \l_@@_transparent_bool
7539        {
7540          \bool_lazy_and:nnF \l_@@_vlines_block_bool \l_@@_hlines_block_bool
7541            {
```

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).

```
7542              \seq_gput_left:Nx \g_@@_pos_of_blocks_seq
7543                { { #1 } { #2 } { #3 } { #4 } { \l_@@_block_name_str } }
7544            }
7545        }


7546      \tl_if_empty:NF \l_@@_draw_tl
7547        {
7548          \bool_lazy_or:nnT \l_@@_hlines_block_bool \l_@@_vlines_block_bool
7549            { \@@_error:n { hlines~with~color } }
7550          \tl_gput_right:Nx \g_nicematrix_code_after_tl
7551            {
7552              \@@_stroke_block:nnn
```

#5 are the options

```
7553              { \exp_not:n { #5 } }
7554              { #1 - #2 }
7555              { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7556            }
7557          \seq_gput_right:Nn \g_@@_pos_of_stroken_blocks_seq
7558            { { #1 } { #2 } { #3 } { #4 } }
7559        }
7560      \clist_if_empty:NF \l_@@_borders_clist
7561        {
7562          \tl_gput_right:Nx \g_nicematrix_code_after_tl
7563            {
7564              \@@_stroke_borders_block:nnn
7565                { \exp_not:n { #5 } }
7566                { #1 - #2 }
7567                { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7568            }
7569        }
7570      \tl_if_empty:NF \l_@@_fill_tl
7571        {
7572          \tl_if_empty:NF \l_@@_opacity_tl
7573            {
7574              \tl_if_head_eq_meaning:nNTF \l_@@_fill_tl [
7575                {
7576                  \tl_set:Nx \l_@@_fill_tl
7577                    {
7578                      [ opacity = \l_@@_opacity_tl ,
7579                        \tl_tail:o \l_@@_fill_tl
7580                    }
7581                }
7582                {
7583                  \tl_set:Nx \l_@@_fill_tl
7584                    { [ opacity = \l_@@_opacity_tl ] { \l_@@_fill_tl } }
7585                }
```

```
7586              }
7587          \tl_gput_right:Nx \g_@@_pre_code_before_tl
7588              {
7589              \exp_not:N \roundedrectanglecolor
7590                \exp_args:No \tl_if_head_eq_meaning:nNTF \l_@@_fill_tl [
7591                  { \l_@@_fill_tl }
7592                  { { \l_@@_fill_tl } }
7593                { #1 - #2 }
7594                { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
7595                { \dim_use:N \l_@@_rounded_corners_dim }
7596              }
7597          }
7598      \seq_if_empty:NF \l_@@_tikz_seq
7599          {
7600          \tl_gput_right:Nx \g_nicematrix_code_before_tl
7601              {
7602              \@@_block_tikz:nnnnn
7603                { #1 }
7604                { #2 }
7605                { \int_use:N \l_@@_last_row_int }
7606                { \int_use:N \l_@@_last_col_int }
7607                { \seq_use:Nn \l_@@_tikz_seq { , } }
7608              }
7609          }

7610      \cs_set_protected_nopar:Npn \diagbox ##1 ##2
7611          {
7612          \tl_gput_right:Nx \g_@@_pre_code_after_tl
7613              {
7614              \@@_actually_diagbox:nnnnnn
7615                { #1 }
7616                { #2 }
7617                { \int_use:N \l_@@_last_row_int }
7618                { \int_use:N \l_@@_last_col_int }
7619                { \exp_not:n { ##1 } } { \exp_not:n { ##2 } }
7620              }
7621          }
```

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.

```
7622        \pgfpicture
7623        \pgfrememberpicturepositiononpagetrue
7624        \pgf@relevantforpicturesizefalse
7625        \@@_qpoint:n { row - #1 }
```

176

```
7626        \dim_set_eq:NN \l_tmpa_dim \pgf@y
7627        \@@_qpoint:n { col - #2 }
7628        \dim_set_eq:NN \l_tmpb_dim \pgf@x
7629        \@@_qpoint:n { row - \int_eval:n { \l_@@_last_row_int + 1 } }
7630        \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7631        \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7632        \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.

```
7633        \@@_pgf_rect_node:nnnnn
7634          { \@@_env: - #1 - #2 - block }
7635          \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
7636        \str_if_empty:NF \l_@@_block_name_str
7637          {
7638            \pgfnodealias
7639              { \@@_env: - \l_@@_block_name_str }
7640              { \@@_env: - #1 - #2 - block }
7641            \str_if_empty:NF \l_@@_name_str
7642              {
7643                \pgfnodealias
7644                  { \l_@@_name_str - \l_@@_block_name_str }
7645                  { \@@_env: - #1 - #2 - block }
7646              }
7647          }
```

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.

```
7648        \bool_if:NF \l_@@_hpos_of_block_cap_bool
7649          {
7650            \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.

```
7651            \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
7652              {
```

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.

```
7653                \cs_if_exist:cT
7654                  { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
7655                  {
7656                    \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
7657                      {
7658                        \pgfpointanchor { \@@_env: - ##1 - #2 } { west }
7659                        \dim_set:Nn \l_tmpb_dim { \dim_min:nn \l_tmpb_dim \pgf@x }
7660                      }
7661                  }
7662              }
```

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.

```
7663            \dim_compare:nNnT \l_tmpb_dim = \c_max_dim
7664              {
7665                \@@_qpoint:n { col - #2 }
7666                \dim_set_eq:NN \l_tmpb_dim \pgf@x
7667              }
7668            \dim_set:Nn \l_@@_tmpd_dim { - \c_max_dim }
7669            \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
7670              {
7671                \cs_if_exist:cT
```

```
7672              { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
7673              {
7674                \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
7675                  {
7676                    \pgfpointanchor
7677                      { \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
7678                      { east }
7679                    \dim_set:Nn \l_@@_tmpd_dim { \dim_max:nn \l_@@_tmpd_dim \pgf@x }
7680                  }
7681              }
7682          }
7683        \dim_compare:nNnT \l_@@_tmpd_dim = { - \c_max_dim }
7684          {
7685            \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
7686            \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
7687          }
7688        \@@_pgf_rect_node:nnnnn
7689          { \@@_env: - #1 - #2 - block - short }
7690          \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
7691      }
```

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.

```
7692        \bool_if:NT \l_@@_medium_nodes_bool
7693          {
7694            \@@_pgf_rect_node:nnn
7695              { \@@_env: - #1 - #2 - block - medium }
7696              { \pgfpointanchor { \@@_env: - #1 - #2 - medium } { north~west } }
7697              {
7698                \pgfpointanchor
7699                  { \@@_env:
7700                    - \int_use:N \l_@@_last_row_int
7701                    - \int_use:N \l_@@_last_col_int - medium
7702                  }
7703                  { south~east }
7704              }
7705          }
7706      \endpgfpicture


7707    \bool_if:NTF \l_@@_ampersand_bool
7708      {
7709        \seq_set_split:Nnn \l_tmpa_seq { & } { #6 }
7710        \int_zero_new:N \l_@@_split_int
7711        \int_set:Nn \l_@@_split_int { \seq_count:N \l_tmpa_seq }
7712        \pgfpicture
7713        \pgfrememberpicturepositiononpagetrue
7714        \pgf@relevantforpicturesizefalse
7715        \@@_qpoint:n { row - #1 }
7716        \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
7717        \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
7718        \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
7719        \@@_qpoint:n { col - #2 }
7720        \dim_set_eq:NN \l_tmpa_dim \pgf@x
7721        \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
7722        \dim_set:Nn \l_tmpb_dim
7723          { ( \pgf@x - \l_tmpa_dim ) / \int_use:N \l_@@_split_int }
7724        \bool_lazy_or:nnT
7725          \l_@@_vlines_block_bool
7726          { \tl_if_eq_p:NN \l_@@_vlines_clist \c_@@_all_tl }
7727          {
7728            \int_step_inline:nn { \l_@@_split_int - 1 }
7729              {
7730                \pgfpathmoveto
```

```
                {
                    \pgfpoint
                      { \l_tmpa_dim + ##1 \l_tmpb_dim }
                      \l_@@_tmpc_dim
                }
              \pgfpathlineto
                {
                    \pgfpoint
                      { \l_tmpa_dim + ##1 \l_tmpb_dim }
                      \l_@@_tmpd_dim
                }
              \CT@arc@
              \pgfsetlinewidth { 1.1 \arrayrulewidth }
              \pgfsetrectcap
              \pgfusepathqstroke
            }
        }
    \@@_qpoint:n { row - #1 - base }
    \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
    \int_step_inline:nn \l_@@_split_int
      {
        \group_begin:
        \dim_set:Nn \col@sep
          { \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
        \pgftransformshift
          {
            \pgfpoint
              {
                \str_case:on \l_@@_hpos_block_str
                  {
                    l { \l_tmpa_dim + ##1 \l_tmpb_dim - \l_tmpb_dim + \col@sep}
                    c { \l_tmpa_dim + ##1 \l_tmpb_dim - 0.5 \l_tmpb_dim }
                    r { \l_tmpa_dim + ##1 \l_tmpb_dim - \col@sep }
                  }
              }
              { \l_@@_tmpc_dim }
          }
        \pgfset
          {
            inner~xsep = \c_zero_dim ,
            inner~ysep = \c_zero_dim
          }
        \pgfnode
          { rectangle }
          {
            \str_case:on \l_@@_hpos_block_str
              {
                c { base }
                l { base~west }
                r { base~east }
              }
          }
          { \seq_item:Nn \l_tmpa_seq { ##1 } } { } { } { }
        \group_end:
      }
    \endpgfpicture
  }
  {
    \bool_if:NTF \l_@@_p_block_bool
      {
```

When the final user has used the key **p**, we have to compute the width.

```
            \pgfpicture
              \pgfrememberpicturepositiononpagetrue
```

179

```
7793              \pgf@relevantforpicturesizefalse
7794              \bool_if:NTF \l_@@_hpos_of_block_cap_bool
7795                {
7796                  \@@_qpoint:n { col - #2 }
7797                  \dim_gset_eq:NN \g_tmpa_dim \pgf@x
7798                  \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } } }
7799                }
7800                {
7801                  \pgfpointanchor { \@@_env: - #1 - #2 - block - short } { west }
7802                  \dim_gset_eq:NN \g_tmpa_dim \pgf@x
7803                  \pgfpointanchor { \@@_env: - #1 - #2 - block - short } { east }
7804                }
7805              \dim_gset:Nn \g_tmpb_dim { \pgf@x - \g_tmpa_dim }
7806            \endpgfpicture
7807            \hbox_set:Nn \l_@@_cell_box
7808              {
7809                \begin { minipage } [ \str_lowercase:o \l_@@_vpos_block_str ]
7810                  { \g_tmpb_dim }
7811                \str_case:on \l_@@_hpos_block_str
7812                  { c \centering r \raggedleft l \raggedright j { } }
7813                #6
7814                \end { minipage }
7815              }
7816          }
7817          { \hbox_set:Nn \l_@@_cell_box { \set@color #6 } }
7818        \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
```

Now, we will put the label of the block.

```
7819        \pgfpicture
7820        \pgfrememberpicturepositiononpagetrue
7821        \pgf@relevantforpicturesizefalse
7822        \bool_lazy_any:nTF
7823          {
7824            { \str_if_eq_p:on \l_@@_vpos_block_str { c } }
7825            { \str_if_eq_p:on \l_@@_vpos_block_str { T } }
7826            { \str_if_eq_p:on \l_@@_vpos_block_str { B } }
7827          }

7828          {
```

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

```
7829            \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.

```
7830            \bool_if:nT \g_@@_last_col_found_bool
7831              {
7832                \int_compare:nNnT { #2 } = \g_@@_col_total_int
7833                  { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_l_str }
7834              }
```

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

```
7835            \tl_set:Nx \l_tmpa_tl
7836              {
7837                \str_case:on \l_@@_vpos_block_str
7838                  {
7839                    c {
7840                        \str_case:on \l_@@_hpos_block_str
7841                          {
7842                            c { center }
7843                            l { west }
7844                            r { east }
7845                            j { center }
7846                          }
7847
```

```
7848                        }
7849                  T {
7850                     \str_case:on \l_@@_hpos_block_str
7851                        {
7852                          c { north }
7853                          l { north~west }
7854                          r { north~east }
7855                          j { north }
7856                        }
7857
7858                  }
7859                  B {
7860                     \str_case:on \l_@@_hpos_block_str
7861                        {
7862                          c { south }
7863                          l { south~west }
7864                          r { south~east }
7865                          j { south }
7866                        }
7867
7868                  }
7869                }
7870              }
7871          \pgftransformshift
7872            {
7873              \pgfpointanchor
7874                {
7875                  \@@_env: - #1 - #2 - block
7876                  \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
7877                }
7878                { \l_tmpa_tl }
7879            }
7880          \pgfset
7881            {
7882              inner~xsep = \c_zero_dim ,
7883              inner~ysep = \c_zero_dim
7884            }
7885          \pgfnode
7886            { rectangle }
7887            { \l_tmpa_tl }
7888            { \box_use_drop:N \l_@@_cell_box } { } { }
7889        }
```

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

```
7890              {
7891              \pgfextracty \l_tmpa_dim
7892                {
7893                  \@@_qpoint:n
7894                    {
7895                      row - \str_if_eq:onTF \l_@@_vpos_block_str { b } { #3 } { #1 }
7896                      - base
7897                    }
7898                }
7899              \dim_sub:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }
```

We retrieve (in \pgf@x) the x-value of the center of the block.

```
7900              \pgfpointanchor
7901                {
7902                  \@@_env: - #1 - #2 - block
7903                  \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
7904                }
7905                {
7906                  \str_case:on \l_@@_hpos_block_str
```

```
7907                    {
7908                      c { center }
7909                      l { west }
7910                      r { east }
7911                      j { center }
7912                    }
7913                  }
```
We put the label of the block which has been composed in `\l_@@_cell_box`.
```
7914                  \pgftransformshift { \pgfpoint \pgf@x \l_tmpa_dim }
7915                  \pgfset { inner~sep = \c_zero_dim }
7916                  \pgfnode
7917                    { rectangle }
7918                    {
7919                      \str_case:on \l_@@_hpos_block_str
7920                        {
7921                          c { base }
7922                          l { base~west }
7923                          r { base~east }
7924                          j { base }
7925                        }
7926                    }
7927                    { \box_use_drop:N \l_@@_cell_box } { } { }
7928                  }
7929              \endpgfpicture
7930            }
7931        \group_end:
7932      }
```

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).
```
7933  \cs_new_protected:Npn \@@_stroke_block:nnn #1 #2 #3
7934    {
7935      \group_begin:
7936      \tl_clear:N \l_@@_draw_tl
7937      \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
7938      \keys_set_known:nn { NiceMatrix / BlockStroke } { #1 }
7939      \pgfpicture
7940      \pgfrememberpicturepositiononpagetrue
7941      \pgf@relevantforpicturesizefalse
7942      \tl_if_empty:NF \l_@@_draw_tl
7943        {
```
If the user has used the key `color` of the command `\Block` without value, the color fixed by `\arrayrulecolor` is used.
```
7944          \tl_if_eq:NNTF \l_@@_draw_tl \c_@@_default_tl
7945            { \CT@arc@ }
7946            { \@@_color:o \l_@@_draw_tl }
7947        }
7948      \pgfsetcornersarced
7949        {
7950          \pgfpoint
7951            { \l_@@_rounded_corners_dim }
7952            { \l_@@_rounded_corners_dim }
7953        }
7954      \@@_cut_on_hyphen:w #2 \q_stop
7955      \int_compare:nNnF \l_tmpa_tl > \c@iRow
7956        {
7957          \int_compare:nNnF \l_tmpb_tl > \c@jCol
7958            {
7959              \@@_qpoint:n { row - \l_tmpa_tl }
7960              \dim_set_eq:NN \l_tmpb_dim \pgf@y
```

```
7961          \@@_qpoint:n { col - \l_tmpb_tl }
7962          \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
7963          \@@_cut_on_hyphen:w #3 \q_stop
7964          \int_compare:nNnT \l_tmpa_tl > \c@iRow
7965            { \tl_set:No \l_tmpa_tl { \int_use:N \c@iRow } }
7966          \int_compare:nNnT \l_tmpb_tl > \c@jCol
7967            { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
7968          \@@_qpoint:n { row - \int_eval:n { \l_tmpa_tl + 1 } }
7969          \dim_set_eq:NN \l_tmpa_dim \pgf@y
7970          \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
7971          \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
7972          \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
7973          \pgfpathrectanglecorners
7974            { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
7975            { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
7976          \dim_compare:nNnTF \l_@@_rounded_corners_dim = \c_zero_dim
7977            { \pgfusepathqstroke }
7978            { \pgfusepath { stroke } }
7979        }
7980      }
7981    \endpgfpicture
7982    \group_end:
7983  }
```

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

```
7984 \keys_define:nn { NiceMatrix / BlockStroke }
7985   {
7986     color .tl_set:N = \l_@@_draw_tl ,
7987     draw .code:n =
7988       \exp_args:Ne \tl_if_empty:nF { #1 } { \tl_set:Nn \l_@@_draw_tl { #1 } } ,
7989     draw .default:n = default ,
7990     line-width .dim_set:N = \l_@@_line_width_dim ,
7991     rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
7992     rounded-corners .default:n = 4 pt
7993   }
```

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).

```
7994 \cs_new_protected:Npn \@@_vlines_block:nnn #1 #2 #3
7995   {
7996     \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
7997     \keys_set_known:nn { NiceMatrix / BlockBorders } { #1 }
7998     \@@_cut_on_hyphen:w #2 \q_stop
7999     \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
8000     \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
8001     \@@_cut_on_hyphen:w #3 \q_stop
8002     \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
8003     \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
8004     \int_step_inline:nnn \l_@@_tmpd_tl \l_tmpb_tl
8005       {
8006         \use:e
8007           {
8008             \@@_vline:n
8009               {
8010                 position = ##1 ,
8011                 start = \l_@@_tmpc_tl ,
8012                 end = \int_eval:n { \l_tmpa_tl - 1 } ,
8013                 total-width = \dim_use:N \l_@@_line_width_dim
8014               }
8015           }
8016       }
8017   }
```

```
8018  \cs_new_protected:Npn \@@_hlines_block:nnn #1 #2 #3
8019    {
8020      \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8021      \keys_set_known:nn { NiceMatrix / BlockBorders } { #1 }
8022      \@@_cut_on_hyphen:w #2 \q_stop
8023      \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
8024      \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
8025      \@@_cut_on_hyphen:w #3 \q_stop
8026      \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
8027      \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
8028      \int_step_inline:nnn \l_@@_tmpc_tl \l_tmpa_tl
8029        {
8030          \use:e
8031            {
8032              \@@_hline:n
8033                {
8034                  position = ##1 ,
8035                  start = \l_@@_tmpd_tl ,
8036                  end = \int_eval:n { \l_tmpb_tl - 1 } ,
8037                  total-width = \dim_use:N \l_@@_line_width_dim
8038                }
8039            }
8040        }
8041    }
```

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).

```
8042  \cs_new_protected:Npn \@@_stroke_borders_block:nnn #1 #2 #3
8043    {
8044      \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
8045      \keys_set_known:nn { NiceMatrix / BlockBorders } { #1 }
8046      \dim_compare:nNnTF \l_@@_rounded_corners_dim > \c_zero_dim
8047        { \@@_error:n { borders~forbidden } }
8048        {
8049          \tl_clear_new:N \l_@@_borders_tikz_tl
8050          \keys_set:nV
8051            { NiceMatrix / OnlyForTikzInBorders }
8052            \l_@@_borders_clist
8053          \@@_cut_on_hyphen:w #2 \q_stop
8054          \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
8055          \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
8056          \@@_cut_on_hyphen:w #3 \q_stop
8057          \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
8058          \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
8059          \@@_stroke_borders_block_i:
8060        }
8061    }

8062  \hook_gput_code:nnn { begindocument } { . }
8063    {
8064      \cs_new_protected:Npx \@@_stroke_borders_block_i:
8065        {
8066          \c_@@_pgfortikzpicture_tl
8067          \@@_stroke_borders_block_ii:
8068          \c_@@_endpgfortikzpicture_tl
8069        }
8070    }

8071  \cs_new_protected:Npn \@@_stroke_borders_block_ii:
8072    {
8073      \pgfremberpicturepositiononpagetrue
8074      \pgf@relevantforpicturesizefalse
8075      \CT@arc@
```

```
8076      \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
8077      \clist_if_in:NnT \l_@@_borders_clist { right }
8078        { \@@_stroke_vertical:n \l_tmpb_tl }
8079      \clist_if_in:NnT \l_@@_borders_clist { left }
8080        { \@@_stroke_vertical:n \l_@@_tmpd_tl }
8081      \clist_if_in:NnT \l_@@_borders_clist { bottom }
8082        { \@@_stroke_horizontal:n \l_tmpa_tl }
8083      \clist_if_in:NnT \l_@@_borders_clist { top }
8084        { \@@_stroke_horizontal:n \l_@@_tmpc_tl }
8085    }
8086  \keys_define:nn { NiceMatrix / OnlyForTikzInBorders }
8087    {
8088      tikz .code:n =
8089        \cs_if_exist:NTF \tikzpicture
8090          { \tl_set:Nn \l_@@_borders_tikz_tl { #1 } }
8091          { \@@_error:n { tikz~in~borders~without~tikz } } ,
8092      tikz .value_required:n = true ,
8093      top .code:n = ,
8094      bottom .code:n = ,
8095      left .code:n = ,
8096      right .code:n = ,
8097      unknown .code:n = \@@_error:n { bad~border }
8098    }
```

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).

```
8099  \cs_new_protected:Npn \@@_stroke_vertical:n #1
8100    {
8101      \@@_qpoint:n \l_@@_tmpc_tl
8102      \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
8103      \@@_qpoint:n \l_tmpa_tl
8104      \dim_set:Nn \l_@@_tmpc_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
8105      \@@_qpoint:n { #1 }
8106      \tl_if_empty:NTF \l_@@_borders_tikz_tl
8107        {
8108          \pgfpathmoveto { \pgfpoint \pgf@x \l_tmpb_dim }
8109          \pgfpathlineto { \pgfpoint \pgf@x \l_@@_tmpc_dim }
8110          \pgfusepathqstroke
8111        }
8112        {
8113          \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
8114            ( \pgf@x , \l_tmpb_dim ) -- ( \pgf@x , \l_@@_tmpc_dim ) ;
8115        }
8116    }
```

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).

```
8117  \cs_new_protected:Npn \@@_stroke_horizontal:n #1
8118    {
8119      \@@_qpoint:n \l_@@_tmpd_tl
8120      \clist_if_in:NnTF \l_@@_borders_clist { left }
8121        { \dim_set:Nn \l_tmpa_dim { \pgf@x - 0.5 \l_@@_line_width_dim } }
8122        { \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \l_@@_line_width_dim } }
8123      \@@_qpoint:n \l_tmpb_tl
8124      \dim_set:Nn \l_tmpb_dim { \pgf@x + 0.5 \l_@@_line_width_dim }
8125      \@@_qpoint:n { #1 }
8126      \tl_if_empty:NTF \l_@@_borders_tikz_tl
8127        {
8128          \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }
8129          \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
8130          \pgfusepathqstroke
8131        }
8132        {
```

```
8133        \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
8134          ( \l_tmpa_dim , \pgf@y ) -- ( \l_tmpb_dim , \pgf@y ) ;
8135      }
8136  }
```

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

```
8137 \keys_define:nn { NiceMatrix / BlockBorders }
8138   {
8139     borders .clist_set:N = \l_@@_borders_clist ,
8140     rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
8141     rounded-corners .default:n = 4 pt ,
8142     line-width .dim_set:N = \l_@@_line_width_dim
8143   }
```

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.

```
8144 \cs_new_protected:Npn \@@_block_tikz:nnnnn #1 #2 #3 #4 #5
8145   {
8146     \begin { tikzpicture }
8147     \@@_clip_with_rounded_corners:
8148     \clist_map_inline:nn { #5 }
8149       {
8150         \keys_set_known:nnN { NiceMatrix / SpecialOffset } { ##1 } \l_tmpa_tl
8151         \use:e { \exp_not:N \path [ \l_tmpa_tl ] }
8152             (
8153               [
8154                 xshift = \dim_use:N \l_@@_offset_dim ,
8155                 yshift = - \dim_use:N \l_@@_offset_dim
8156               ]
8157               #1 -| #2
8158             )
8159             rectangle
8160             (
8161               [
8162                 xshift = - \dim_use:N \l_@@_offset_dim ,
8163                 yshift = \dim_use:N \l_@@_offset_dim
8164               ]
8165               \int_eval:n { #3 + 1 } -| \int_eval:n { #4 + 1 }
8166             ) ;
8167       }
8168     \end { tikzpicture }
8169   }
8170 \cs_generate_variant:Nn \@@_block_tikz:nnnnn { n n n n V }
```

```
8171 \keys_define:nn { NiceMatrix / SpecialOffset }
8172   { offset .dim_set:N = \l_@@_offset_dim }
```

28   How to draw the dotted lines transparently

```
8173 \cs_set_protected:Npn \@@_renew_matrix:
8174   {
8175     \RenewDocumentEnvironment { pmatrix } { }
8176       { \pNiceMatrix }
8177       { \endpNiceMatrix }
8178     \RenewDocumentEnvironment { vmatrix } { }
8179       { \vNiceMatrix }
```

```
8180        { \endvNiceMatrix }
8181      \RenewDocumentEnvironment { Vmatrix } { }
8182        { \VNiceMatrix }
8183        { \endVNiceMatrix }
8184      \RenewDocumentEnvironment { bmatrix } { }
8185        { \bNiceMatrix }
8186        { \endbNiceMatrix }
8187      \RenewDocumentEnvironment { Bmatrix } { }
8188        { \BNiceMatrix }
8189        { \endBNiceMatrix }
8190    }
```

29   Automatic arrays

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

```
8191  \keys_define:nn { NiceMatrix / Auto }
8192    {
8193      columns-type .tl_set:N = \l_@@_columns_type_tl ,
8194      columns-type .value_required:n = true ,
8195      l .meta:n = { columns-type = l } ,
8196      r .meta:n = { columns-type = r } ,
8197      c .meta:n = { columns-type = c } ,
8198      delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8199      delimiters / color .value_required:n = true ,
8200      delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
8201      delimiters / max-width .default:n = true ,
8202      delimiters .code:n = \keys_set:nn { NiceMatrix / delimiters } { #1 } ,
8203      delimiters .value_required:n = true ,
8204      rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
8205      rounded-corners .default:n = 4 pt
8206    }
8207  \NewDocumentCommand \AutoNiceMatrixWithDelims
8208    { m m O { } > { \SplitArgument { 1 } { - } } m O { } m ! O { } }
8209    { \@@_auto_nice_matrix:nnnnnn { #1 } { #2 } #4 { #6 } { #3 , #5 , #7  } }
8210  \cs_new_protected:Npn \@@_auto_nice_matrix:nnnnnn #1 #2 #3 #4 #5 #6
8211    {
```

The group is for the protection of the keys.

```
8212      \group_begin:
8213      \keys_set_known:nnN { NiceMatrix / Auto } { #6 } \l_tmpa_tl
8214      \use:e
8215        {
8216          \exp_not:N \begin { NiceArrayWithDelims } { #1 } { #2 }
8217            { * { #4 } { \exp_not:o \l_@@_columns_type_tl } }
8218            [ \exp_not:o \l_tmpa_tl ]
8219        }
8220      \int_if_zero:nT \l_@@_first_row_int
8221        {
8222          \int_if_zero:nT \l_@@_first_col_int { & }
8223          \prg_replicate:nn { #4 - 1 } { & }
8224          \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
8225        }
8226      \prg_replicate:nn { #3 }
8227        {
8228          \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).

```
8229          \prg_replicate:nn { #4 - 1 } { { } #5 & } #5
8230          \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
```

```
8231            }
8232        \int_compare:nNnT \l_@@_last_row_int > { -2 }
8233          {
8234            \int_if_zero:nT \l_@@_first_col_int { & }
8235            \prg_replicate:nn { #4 - 1 } { & }
8236            \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
8237          }
8238        \end { NiceArrayWithDelims }
8239        \group_end:
8240      }
8241  \cs_set_protected:Npn \@@_define_com:nnn #1 #2 #3
8242    {
8243      \cs_set_protected:cpn { #1 AutoNiceMatrix }
8244        {
8245          \bool_gset_true:N \g_@@_delims_bool
8246          \str_gset:Nx \g_@@_name_env_str { #1 AutoNiceMatrix }
8247          \AutoNiceMatrixWithDelims { #2 } { #3 }
8248        }
8249    }
8250  \@@_define_com:nnn p ( )
8251  \@@_define_com:nnn b [ ]
8252  \@@_define_com:nnn v | |
8253  \@@_define_com:nnn V \| \|
8254  \@@_define_com:nnn B \{ \}
```

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

```
8255  \NewDocumentCommand \AutoNiceMatrix { O { } m O { } m ! O { } }
8256    {
8257      \group_begin:
8258      \bool_gset_false:N \g_@@_delims_bool
8259      \AutoNiceMatrixWithDelims . . { #2 } { #4 } [ #1 , #3 , #5 ]
8260      \group_end:
8261    }
```

30   The redefinition of the command \dotfill

```
8262  \cs_set_eq:NN \@@_old_dotfill \dotfill
8263  \cs_new_protected:Npn \@@_dotfill:
8264    {
```

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}`).

```
8265      \@@_old_dotfill
8266      \tl_gput_right:Nn \g_@@_cell_after_hook_tl \@@_dotfill_i:
8267    }
```

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`.

```
8268  \cs_new_protected:Npn \@@_dotfill_i:
8269    { \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.

```
8270  \cs_new_protected:Npn \@@_diagbox:nn #1 #2
8271    {
8272      \tl_gput_right:Nx \g_@@_pre_code_after_tl
8273        {
8274          \@@_actually_diagbox:nnnnnn
8275            { \int_use:N \c@iRow }
8276            { \int_use:N \c@jCol }
8277            { \int_use:N \c@iRow }
8278            { \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.

```
8279          { \g_@@_row_style_tl \exp_not:n { #1 } }
8280          { \g_@@_row_style_tl \exp_not:n { #2 } }
8281        }
```

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.

```
8282      \seq_gput_right:Nx \g_@@_pos_of_blocks_seq
8283        {
8284          { \int_use:N \c@iRow }
8285          { \int_use:N \c@jCol }
8286          { \int_use:N \c@iRow }
8287          { \int_use:N \c@jCol }
```

The last argument is for the name of the block.

```
8288          { }
8289        }
8290    }
```

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.

```
8291  \cs_new_protected:Npn \@@_actually_diagbox:nnnnnn #1 #2 #3 #4 #5 #6
8292    {
8293      \pgfpicture
8294      \pgf@relevantforpicturesizefalse
8295      \pgfrememberpicturepositiononpagetrue
8296      \@@_qpoint:n { row - #1 }
8297      \dim_set_eq:NN \l_tmpa_dim \pgf@y
8298      \@@_qpoint:n { col - #2 }
8299      \dim_set_eq:NN \l_tmpb_dim \pgf@x
8300      \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
8301      \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
8302      \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
8303      \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
8304      \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
8305      \pgfpathlineto { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
8306        {
```

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.

```
8307          \CT@arc@
8308          \pgfsetroundcap
8309          \pgfusepathqstroke
8310        }
8311      \pgfset { inner~sep = 1 pt }
8312      \pgfscope
8313      \pgftransformshift { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
```

```
8314      \pgfnode { rectangle } { south~west }
8315        {
8316          \begin { minipage } { 20 cm }
8317          \@@_math_toggle: #5 \@@_math_toggle:
8318          \end { minipage }
8319        }
8320        { }
8321        { }
8322      \endpgfscope
8323      \pgftransformshift { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
8324      \pgfnode { rectangle } { north~east }
8325        {
8326          \begin { minipage } { 20 cm }
8327          \raggedleft
8328          \@@_math_toggle: #6 \@@_math_toggle:
8329          \end { minipage }
8330        }
8331        { }
8332        { }
8333      \endpgfpicture
8334    }
```

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.

```
8335 \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 \\.

```
8336 \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.

```
8337 \cs_new_protected:Npn \@@_CodeAfter_ii:n #1 \end
8338   {
8339     \tl_gput_right:Nn \g_nicematrix_code_after_tl { #1 }
8340     \@@_CodeAfter_iv:n
8341   }
```

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

```
8342 \cs_new_protected:Npn \@@_CodeAfter_iv:n #1
8343   {
```

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.

```
8344     \str_if_eq:eeTF \@currenvir { #1 }
8345       { \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.

```
8346       {
8347         \tl_gput_right:Nn \g_nicematrix_code_after_tl { \end { #1 } }
8348         \@@_CodeAfter_ii:n
8349       }
8350   }
```

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.

```
8351  \cs_new_protected:Npn \@@_delimiter:nnn #1 #2 #3
8352    {
8353      \pgfpicture
8354      \pgfrememberpicturepositiononpagetrue
8355      \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.

```
8356      \@@_qpoint:n { row - 1 }
8357      \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
8358      \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
8359      \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).

```
8360      \bool_if:nTF { #3 }
8361        { \dim_set_eq:NN \l_tmpa_dim \c_max_dim }
8362        { \dim_set:Nn \l_tmpa_dim { - \c_max_dim } }
8363      \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
8364        {
8365          \cs_if_exist:cT
8366            { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
8367            {
8368              \pgfpointanchor
8369                { \@@_env: - ##1 - #2 }
8370                { \bool_if:nTF { #3 } { west } { east } }
8371              \dim_set:Nn \l_tmpa_dim
8372                { \bool_if:nTF { #3 } \dim_min:nn \dim_max:nn \l_tmpa_dim \pgf@x }
8373            }
8374        }
```

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

```
8375      \pgfset { inner~sep = \c_zero_dim }
8376      \dim_zero:N \nulldelimiterspace
8377      \pgftransformshift
8378        {
8379          \pgfpoint
8380            { \l_tmpa_dim }
8381            { ( \l_@@_y_initial_dim + \l_@@_y_final_dim + \arrayrulewidth ) / 2 }
8382        }
8383      \pgfnode
8384        { rectangle }
8385        { \bool_if:nTF { #3 } { east } { west } }
8386        {
```

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

```
8387          \nullfont
8388          \c_math_toggle_token
8389          \@@_color:o \l_@@_delimiters_color_tl
8390          \bool_if:nTF { #3 } { \left #1 } { \left . }
```

```
8391          \vcenter
8392            {
8393              \nullfont
8394              \hrule \@height
8395                    \dim_eval:n { \l_@@_y_initial_dim - \l_@@_y_final_dim }
8396                    \@depth \c_zero_dim
8397                    \@width \c_zero_dim
8398            }
8399          \bool_if:nTF { #3 } { \right . } { \right #1 }
8400          \c_math_toggle_token
8401        }
8402        { }
8403        { }
8404      \endpgfpicture
8405    }
```

34   The command \SubMatrix

```
8406 \keys_define:nn { NiceMatrix / sub-matrix }
8407    {
8408      extra-height .dim_set:N = \l_@@_submatrix_extra_height_dim ,
8409      extra-height .value_required:n = true ,
8410      left-xshift .dim_set:N = \l_@@_submatrix_left_xshift_dim ,
8411      left-xshift .value_required:n = true ,
8412      right-xshift .dim_set:N = \l_@@_submatrix_right_xshift_dim ,
8413      right-xshift .value_required:n = true ,
8414      xshift .meta:n = { left-xshift = #1, right-xshift = #1 } ,
8415      xshift .value_required:n = true ,
8416      delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8417      delimiters / color .value_required:n = true ,
8418      slim .bool_set:N = \l_@@_submatrix_slim_bool ,
8419      slim .default:n = true ,
8420      hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
8421      hlines .default:n = all ,
8422      vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
8423      vlines .default:n = all ,
8424      hvlines .meta:n = { hlines, vlines } ,
8425      hvlines .value_forbidden:n = true
8426    }
8427 \keys_define:nn { NiceMatrix }
8428    {
8429      SubMatrix .inherit:n = NiceMatrix / sub-matrix ,
8430      NiceArray / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
8431      pNiceArray / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
8432      NiceMatrixOptions / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
8433    }
```

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

```
8434 \keys_define:nn { NiceMatrix / SubMatrix }
8435    {
8436      delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
8437      delimiters / color .value_required:n = true ,
8438      hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
8439      hlines .default:n = all ,
8440      vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
8441      vlines .default:n = all ,
8442      hvlines .meta:n = { hlines, vlines } ,
8443      hvlines .value_forbidden:n = true ,
8444      name .code:n =
```

```
8445        \tl_if_empty:nTF { #1 }
8446          { \@@_error:n { Invalid~name } }
8447          {
8448            \regex_match:nnTF { \A[A-Za-z][A-Za-z0-9]*\Z } { #1 }
8449              {
8450                \seq_if_in:NnTF \g_@@_submatrix_names_seq { #1 }
8451                  { \@@_error:nn { Duplicate~name~for~SubMatrix } { #1 } }
8452                  {
8453                    \str_set:Nn \l_@@_submatrix_name_str { #1 }
8454                    \seq_gput_right:Nn \g_@@_submatrix_names_seq { #1 }
8455                  }
8456              }
8457              { \@@_error:n { Invalid~name } }
8458          } ,
8459      name .value_required:n = true ,
8460      rules .code:n = \keys_set:nn { NiceMatrix / rules } { #1 } ,
8461      rules .value_required:n = true ,
8462      code .tl_set:N = \l_@@_code_tl ,
8463      code .value_required:n = true ,
8464      unknown .code:n = \@@_error:n { Unknown~key~for~SubMatrix }
8465    }


8466  \NewDocumentCommand \@@_SubMatrix_in_code_before { m m m m ! O { } }
8467    {
8468      \peek_remove_spaces:n
8469        {
8470          \tl_gput_right:Nx \g_@@_pre_code_after_tl
8471            {
8472              \SubMatrix { #1 } { #2 } { #3 } { #4 }
8473                [
8474                  delimiters / color = \l_@@_delimiters_color_tl ,
8475                  hlines = \l_@@_submatrix_hlines_clist ,
8476                  vlines = \l_@@_submatrix_vlines_clist ,
8477                  extra-height = \dim_use:N \l_@@_submatrix_extra_height_dim ,
8478                  left-xshift = \dim_use:N \l_@@_submatrix_left_xshift_dim ,
8479                  right-xshift = \dim_use:N \l_@@_submatrix_right_xshift_dim ,
8480                  slim = \bool_to_str:N \l_@@_submatrix_slim_bool ,
8481                  #5
8482                ]
8483            }
8484          \@@_SubMatrix_in_code_before_i { #2 } { #3 }
8485        }
8486    }

8487  \NewDocumentCommand \@@_SubMatrix_in_code_before_i
8488    { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
8489    { \@@_SubMatrix_in_code_before_i:nnnn #1 #2 }

8490  \cs_new_protected:Npn \@@_SubMatrix_in_code_before_i:nnnn #1 #2 #3 #4
8491    {
8492      \seq_gput_right:Nx \g_@@_submatrix_seq
8493        {
```

We use `\str_if_eq:nnTF` because it is fully expandable.

```
8494          { \str_if_eq:nnTF { #1 } { last } { \int_use:N \c@iRow } { #1 } }
8495          { \str_if_eq:nnTF { #2 } { last } { \int_use:N \c@jCol } { #2 } }
8496          { \str_if_eq:nnTF { #3 } { last } { \int_use:N \c@iRow } { #3 } }
8497          { \str_if_eq:nnTF { #4 } { last } { \int_use:N \c@jCol } { #4 } }
8498        }
8499    }
```

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`.

```
8500 \hook_gput_code:nnn { begindocument } { . }
8501   {
8502     \cs_set_nopar:Npn \l_@@_argspec_tl { m m m m O { } E { _ ^ } { { } { } { } } }
8503     \tl_set_rescan:Nno  \l_@@_argspec_tl { } \l_@@_argspec_tl
8504     \exp_args:NNo \NewDocumentCommand \@@_SubMatrix \l_@@_argspec_tl
8505       {
8506         \peek_remove_spaces:n
8507           {
8508             \@@_sub_matrix:nnnnnnn
8509               { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } { #7 }
8510           }
8511       }
8512   }
```

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`).

```
8513 \NewDocumentCommand \@@_compute_i_j:nn
8514   { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
8515   { \@@_compute_i_j:nnnn #1 #2 }
8516 \cs_new_protected:Npn \@@_compute_i_j:nnnn #1 #2 #3 #4
8517   {
8518     \cs_set_nopar:Npn \l_@@_first_i_tl { #1 }
8519     \cs_set_nopar:Npn \l_@@_first_j_tl { #2 }
8520     \cs_set_nopar:Npn \l_@@_last_i_tl { #3 }
8521     \cs_set_nopar:Npn \l_@@_last_j_tl { #4 }
8522     \tl_if_eq:NnT \l_@@_first_i_tl { last }
8523       { \tl_set:NV \l_@@_first_i_tl \c@iRow }
8524     \tl_if_eq:NnT \l_@@_first_j_tl { last }
8525       { \tl_set:NV \l_@@_first_j_tl \c@jCol }
8526     \tl_if_eq:NnT \l_@@_last_i_tl { last }
8527       { \tl_set:NV \l_@@_last_i_tl \c@iRow }
8528     \tl_if_eq:NnT \l_@@_last_j_tl { last }
8529       { \tl_set:NV \l_@@_last_j_tl \c@jCol }
8530   }
8531 \cs_new_protected:Npn \@@_sub_matrix:nnnnnnn #1 #2 #3 #4 #5 #6 #7
8532   {
8533     \group_begin:
```

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

```
8534     \@@_compute_i_j:nn { #2 } { #3 }
8535     \int_compare:nNnT \l_@@_first_i_tl = \l_@@_last_i_tl
8536       { \cs_set_nopar:Npn \arraystretch { 1 } }
8537     \bool_lazy_or:nnTF
8538       { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
8539       { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
8540       { \@@_error:nn { Construct~too~large } { \SubMatrix } }
8541       {
8542         \str_clear_new:N \l_@@_submatrix_name_str
8543         \keys_set:nn { NiceMatrix / SubMatrix } { #5 }
```

```
8544        \pgfpicture
8545        \pgfrememberpicturepositiononpagetrue
8546        \pgf@relevantforpicturesizefalse
8547        \pgfset { inner~sep = \c_zero_dim }
8548        \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
8549        \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
```

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

```
8550        \bool_if:NTF \l_@@_submatrix_slim_bool
8551          { \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl }
8552          { \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int }
8553          {
8554            \cs_if_exist:cT
8555              { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
8556              {
8557                \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
8558                \dim_set:Nn \l_@@_x_initial_dim
8559                  { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
8560              }
8561            \cs_if_exist:cT
8562              { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
8563              {
8564                \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
8565                \dim_set:Nn \l_@@_x_final_dim
8566                  { \dim_max:nn \l_@@_x_final_dim \pgf@x }
8567              }
8568          }
8569        \dim_compare:nNnTF \l_@@_x_initial_dim = \c_max_dim
8570          { \@@_error:nn { Impossible~delimiter } { left } }
8571          {
8572            \dim_compare:nNnTF \l_@@_x_final_dim = { - \c_max_dim }
8573              { \@@_error:nn { Impossible~delimiter } { right } }
8574              { \@@_sub_matrix_i:nnnn { #1 } { #4 } { #6 } { #7 } }
8575          }
8576        \endpgfpicture
8577      }
8578    \group_end:
8579  }
```

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

```
8580 \cs_new_protected:Npn \@@_sub_matrix_i:nnnn #1 #2 #3 #4
8581   {
8582     \@@_qpoint:n { row - \l_@@_first_i_tl - base }
8583     \dim_set:Nn \l_@@_y_initial_dim
8584       {
8585         \fp_to_dim:n
8586           {
8587             \pgf@y
8588             + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
8589           }
8590       }
8591     \@@_qpoint:n { row - \l_@@_last_i_tl - base }
8592     \dim_set:Nn \l_@@_y_final_dim
8593       { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
8594     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
8595       {
8596         \cs_if_exist:cT
8597           { pgf @ sh @ ns @ \@@_env: - \l_@@_first_i_tl - ##1 }
8598           {
8599             \pgfpointanchor { \@@_env: - \l_@@_first_i_tl - ##1 } { north }
8600             \dim_set:Nn \l_@@_y_initial_dim
8601               { \dim_max:nn \l_@@_y_initial_dim \pgf@y }
8602           }
```

```
8603        \cs_if_exist:cT
8604          { pgf @ sh @ ns @ \@@_env: - \l_@@_last_i_tl - ##1 }
8605          {
8606            \pgfpointanchor { \@@_env: - \l_@@_last_i_tl - ##1 } { south }
8607            \dim_set:Nn \l_@@_y_final_dim
8608              { \dim_min:nn \l_@@_y_final_dim \pgf@y }
8609          }
8610        }
8611      \dim_set:Nn \l_tmpa_dim
8612        {
8613          \l_@@_y_initial_dim - \l_@@_y_final_dim +
8614          \l_@@_submatrix_extra_height_dim - \arrayrulewidth
8615        }
8616      \dim_zero:N \nulldelimiterspace
```

We will draw the rules in the \SubMatrix.

```
8617        \group_begin:
8618        \pgfsetlinewidth { 1.1 \arrayrulewidth }
8619        \@@_set_CT@arc@:o \l_@@_rules_color_tl
8620        \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.

```
8621        \seq_map_inline:Nn \g_@@_cols_vlism_seq
8622          {
8623            \int_compare:nNnT \l_@@_first_j_tl < { ##1 }
8624              {
8625                \int_compare:nNnT
8626                  { ##1 } < { \int_eval:n { \l_@@_last_j_tl + 1 } }
8627                  {
```

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

```
8628                    \@@_qpoint:n { col - ##1 }
8629                    \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
8630                    \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
8631                    \pgfusepathqstroke
8632                  }
8633              }
8634          }
```

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.

```
8635        \tl_if_eq:NNTF \l_@@_submatrix_vlines_clist \c_@@_all_tl
8636          { \int_step_inline:nn { \l_@@_last_j_tl - \l_@@_first_j_tl } }
8637          { \clist_map_inline:Nn \l_@@_submatrix_vlines_clist }
8638          {
8639            \bool_lazy_and:nnTF
8640              { \int_compare_p:nNn { ##1 } > \c_zero_int }
8641              {
8642                \int_compare_p:nNn
8643                  { ##1 } < { \l_@@_last_j_tl - \l_@@_first_j_tl + 1 } }
8644              {
8645                \@@_qpoint:n { col - \int_eval:n { ##1 + \l_@@_first_j_tl } }
8646                \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
8647                \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
8648                \pgfusepathqstroke
8649              }
8650              { \@@_error:nnn { Wrong~line~in~SubMatrix } { vertical } { ##1 } }
8651          }
```

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.

```
8652        \tl_if_eq:NNTF \l_@@_submatrix_hlines_clist \c_@@_all_tl
8653          { \int_step_inline:nn { \l_@@_last_i_tl - \l_@@_first_i_tl } }
8654          { \clist_map_inline:Nn \l_@@_submatrix_hlines_clist }
8655          {
8656            \bool_lazy_and:nnTF
8657              { \int_compare_p:nNn { ##1 } > \c_zero_int }
8658              {
8659                \int_compare_p:nNn
8660                  { ##1 } < { \l_@@_last_i_tl - \l_@@_first_i_tl + 1 } }
8661              {
8662                \@@_qpoint:n { row - \int_eval:n { ##1 + \l_@@_first_i_tl } }
```
We use a group to protect \l_tmpa_dim and \l_tmpb_dim.
```
8663                \group_begin:
```
We compute in \l_tmpa_dim the x-value of the left end of the rule.
```
8664                \dim_set:Nn \l_tmpa_dim
8665                  { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
8666                \str_case:nn { #1 }
8667                  {
8668                    (  { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
8669                    [  { \dim_sub:Nn \l_tmpa_dim { 0.2 mm } }
8670                    \{ { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
8671                  }
8672                \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }
```
We compute in \l_tmpb_dim the x-value of the right end of the rule.
```
8673                \dim_set:Nn \l_tmpb_dim
8674                  { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
8675                \str_case:nn { #2 }
8676                  {
8677                    )  { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
8678                    ]  { \dim_add:Nn \l_tmpb_dim { 0.2 mm } }
8679                    \} { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
8680                  }
8681                \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
8682                \pgfusepathqstroke
8683                \group_end:
8684              }
8685            { \@@_error:nnn { Wrong~line~in~SubMatrix } { horizontal } { ##1 } }
8686          }
```
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).
```
8687        \str_if_empty:NF \l_@@_submatrix_name_str
8688          {
8689            \@@_pgf_rect_node:nnnnn \l_@@_submatrix_name_str
8690              \l_@@_x_initial_dim \l_@@_y_initial_dim
8691              \l_@@_x_final_dim \l_@@_y_final_dim
8692          }
8693        \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`.
```
8694        \begin { pgfscope }
8695        \pgftransformshift
8696          {
8697            \pgfpoint
8698              { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
8699              { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
8700          }
8701        \str_if_empty:NTF \l_@@_submatrix_name_str
8702          { \@@_node_left:nn #1 { } }
```

```
8703        { \@@_node_left:nn #1 { \@@_env: - \l_@@_submatrix_name_str - left } }
8704      \end { pgfscope }
```

Now, we deal with the right delimiter.

```
8705      \pgftransformshift
8706        {
8707          \pgfpoint
8708            { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
8709            { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
8710        }
8711      \str_if_empty:NTF \l_@@_submatrix_name_str
8712        { \@@_node_right:nnnn #2 { } { #3 } { #4 } }
8713        {
8714          \@@_node_right:nnnn #2
8715            { \@@_env: - \l_@@_submatrix_name_str - right } { #3 } { #4 }
8716        }
8717      \cs_set_eq:NN \pgfpointanchor \@@_pgfpointanchor:n
8718      \flag_clear_new:n { nicematrix }
8719      \l_@@_code_tl
8720    }
```

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 columm *relative* of the current `\SubMatrix`. That's why we will patch (locally in the `\SubMatrix`) the command `\pgfpointanchor`.

```
8721 \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.

```
8722 \cs_new_protected:Npn \@@_pgfpointanchor:n #1
8723   {
8724     \use:e
8725       { \exp_not:N \@@_old_pgfpointanchor { \@@_pgfpointanchor_i:nn #1 } }
8726   }
```

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 -.

```
8727 \cs_new:Npn \@@_pgfpointanchor_i:nn #1 #2
8728   { #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.

```
8729 \tl_const:Nn \c_@@_integers_alist_tl
8730   {
8731     { 1 } { } { 2 } { } { 3 } { } { 4 } { } { 5 } { }
8732     { 6 } { } { 7 } { } { 8 } { } { 9 } { } { 10 } { }
8733     { 11 } { } { 12 } { } { 13 } { } { 14 } { } { 15 } { }
8734     { 16 } { } { 17 } { } { 18 } { } { 19 } { } { 20 } { }
8735   }
```

```
8736 \cs_new:Npn \@@_pgfpointanchor_ii:w #1-#2\q_stop
8737   {
```

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`.

```
8738      \tl_if_empty:nTF { #2 }
8739        {
8740          \str_case:nVTF { #1 } \c_@@_integers_alist_tl
8741            {
8742              \flag_raise:n { nicematrix }
8743              \int_if_even:nTF { \flag_height:n { nicematrix } }
8744                { \int_eval:n { #1 + \l_@@_first_i_tl - 1 } }
8745                { \int_eval:n { #1 + \l_@@_first_j_tl - 1 } }
8746            }
8747            { #1 }
8748        }
```

If there is an hyphen, we have to see whether we have a node of the form i-j, `row`-i or `col`-j.

```
8749        { \@@_pgfpointanchor_iii:w { #1 } #2 }
8750    }
```

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`).

```
8751 \cs_new:Npn \@@_pgfpointanchor_iii:w #1 #2 -
8752    {
8753      \str_case:nnF { #1 }
8754        {
8755          { row } { row - \int_eval:n { #2 + \l_@@_first_i_tl - 1 } }
8756          { col } { col - \int_eval:n { #2 + \l_@@_first_j_tl - 1 } }
8757        }
```

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

```
8758        {
8759          \int_eval:n { #1 + \l_@@_first_i_tl - 1 }
8760          - \int_eval:n { #2 + \l_@@_first_j_tl - 1 }
8761        }
8762    }
```

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`).

```
8763 \cs_new_protected:Npn \@@_node_left:nn #1 #2
8764    {
8765      \pgfnode
8766        { rectangle }
8767        { east }
8768        {
8769          \nullfont
8770          \c_math_toggle_token
8771          \@@_color:o \l_@@_delimiters_color_tl
8772          \left #1
8773          \vcenter
8774            {
8775              \nullfont
8776              \hrule \@height \l_tmpa_dim
8777                     \@depth \c_zero_dim
8778                     \@width \c_zero_dim
8779            }
8780          \right .
8781          \c_math_toggle_token
8782        }
8783        { #2 }
```

```
8784        { }
8785    }
```

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.

```
8786 \cs_new_protected:Npn \@@_node_right:nnnn #1 #2 #3 #4
8787    {
8788      \pgfnode
8789        { rectangle }
8790        { west }
8791        {
8792          \nullfont
8793          \c_math_toggle_token
8794          \@@_color:o \l_@@_delimiters_color_tl
8795          \left .
8796          \vcenter
8797            {
8798              \nullfont
8799              \hrule \@height \l_tmpa_dim
8800                     \@depth \c_zero_dim
8801                     \@width \c_zero_dim
8802            }
8803          \right #1
8804          \tl_if_empty:nF { #3 } { _ { \smash { #3 } } }
8805          ^ { \smash { #4 } }
8806          \c_math_toggle_token
8807        }
8808        { #2 }
8809        { }
8810    }
```

35   Les commandes \UnderBrace et \OverBrace

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

```
8811 \NewDocumentCommand \@@_UnderBrace { O { } m m m O { } }
8812    {
8813      \peek_remove_spaces:n
8814        { \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { under } }
8815    }
8816 \NewDocumentCommand \@@_OverBrace { O { } m m m O { } }
8817    {
8818      \peek_remove_spaces:n
8819        { \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { over } }
8820    }

8821 \keys_define:nn { NiceMatrix / Brace }
8822    {
8823      left-shorten .bool_set:N = \l_@@_brace_left_shorten_bool ,
8824      left-shorten .default:n = true ,
8825      right-shorten .bool_set:N = \l_@@_brace_right_shorten_bool ,
8826      shorten .meta:n = { left-shorten , right-shorten } ,
8827      right-shorten .default:n = true ,
8828      yshift .dim_set:N = \l_@@_brace_yshift_dim ,
8829      yshift .value_required:n = true ,
8830      yshift .initial:n = \c_zero_dim ,
8831      color .tl_set:N = \l_tmpa_tl ,
8832      color .value_required:n = true ,
```

```
8833        unknown .code:n = \@@_error:n { Unknown~key~for~Brace }
8834    }
```

#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.

```
8835    \cs_new_protected:Npn \@@_brace:nnnnn #1 #2 #3 #4 #5
8836      {
8837        \group_begin:
```

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

```
8838        \@@_compute_i_j:nn { #1 } { #2 }
8839        \bool_lazy_or:nnTF
8840          { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
8841          { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
8842          {
8843            \str_if_eq:nnTF { #5 } { under }
8844              { \@@_error:nn { Construct~too~large } { \UnderBrace } }
8845              { \@@_error:nn { Construct~too~large } { \OverBrace } }
8846          }
8847          {
8848            \tl_clear:N \l_tmpa_tl
8849            \keys_set:nn { NiceMatrix / Brace } { #4 }
8850            \tl_if_empty:NF \l_tmpa_tl { \color { \l_tmpa_tl } }
8851            \pgfpicture
8852            \pgfrememberpicturepositiononpagetrue
8853            \pgf@relevantforpicturesizefalse
8854            \bool_if:NT \l_@@_brace_left_shorten_bool
8855              {
8856                \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
8857                \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
8858                  {
8859                    \cs_if_exist:cT
8860                      { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
8861                      {
8862                        \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
8863                        \dim_set:Nn \l_@@_x_initial_dim
8864                          { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
8865                      }
8866                  }
8867              }
8868            \bool_lazy_or:nnT
8869              { \bool_not_p:n \l_@@_brace_left_shorten_bool }
8870              { \dim_compare_p:nNn \l_@@_x_initial_dim = \c_max_dim }
8871              {
8872                \@@_qpoint:n { col - \l_@@_first_j_tl }
8873                \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
8874              }
8875            \bool_if:NT \l_@@_brace_right_shorten_bool
8876              {
8877                \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
8878                \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
8879                  {
8880                    \cs_if_exist:cT
8881                      { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
8882                      {
8883                        \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
8884                        \dim_set:Nn \l_@@_x_final_dim
8885                          { \dim_max:nn \l_@@_x_final_dim \pgf@x }
8886                      }
8887                  }
8888              }
8889            \bool_lazy_or:nnT
8890              { \bool_not_p:n \l_@@_brace_right_shorten_bool }
```

```
8891          { \dim_compare_p:nNn \l_@@_x_final_dim = { - \c_max_dim } }
8892          {
8893            \@@_qpoint:n { col - \int_eval:n { \l_@@_last_j_tl + 1 } }
8894            \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
8895          }
8896        \pgfset { inner~sep = \c_zero_dim }
8897        \str_if_eq:nnTF { #5 } { under }
8898          { \@@_underbrace_i:n { #3 } }
8899          { \@@_overbrace_i:n { #3 } }
8900        \endpgfpicture
8901      }
8902    \group_end:
8903  }
```

The argument is the text to put above the brace.

```
8904 \cs_new_protected:Npn \@@_overbrace_i:n #1
8905   {
8906     \@@_qpoint:n { row - \l_@@_first_i_tl }
8907     \pgftransformshift
8908       {
8909         \pgfpoint
8910         { ( \l_@@_x_initial_dim + \l_@@_x_final_dim) / 2 }
8911         { \pgf@y + \l_@@_brace_yshift_dim - 3 pt}
8912       }
8913     \pgfnode
8914       { rectangle }
8915       { south }
8916       {
8917         \vtop
8918           {
8919             \group_begin:
8920             \everycr { }
8921             \halign
8922               {
8923                 \hfil ## \hfil \crcr
8924                 \@@_math_toggle: #1 \@@_math_toggle: \cr
8925                 \noalign { \skip_vertical:n { 3 pt } \nointerlineskip }
8926                 \c_math_toggle_token
8927                 \overbrace
8928                   {
8929                     \hbox_to_wd:nn
8930                       { \l_@@_x_final_dim - \l_@@_x_initial_dim }
8931                       { }
8932                   }
8933                 \c_math_toggle_token
8934               \cr
8935               }
8936             \group_end:
8937           }
8938       }
8939       { }
8940       { }
8941   }
```

The argument is the text to put under the brace.

```
8942 \cs_new_protected:Npn \@@_underbrace_i:n #1
8943   {
8944     \@@_qpoint:n { row - \int_eval:n { \l_@@_last_i_tl + 1 } }
8945     \pgftransformshift
8946       {
8947         \pgfpoint
8948         { ( \l_@@_x_initial_dim + \l_@@_x_final_dim) / 2 }
8949         { \pgf@y  - \l_@@_brace_yshift_dim + 3 pt }
```

```
8950          }
8951        \pgfnode
8952          { rectangle }
8953          { north }
8954          {
8955            \group_begin:
8956            \everycr { }
8957            \vbox
8958              {
8959                \halign
8960                  {
8961                    \hfil ## \hfil \crcr
8962                    \c_math_toggle_token
8963                    \underbrace
8964                      {
8965                        \hbox_to_wd:nn
8966                          { \l_@@_x_final_dim - \l_@@_x_initial_dim }
8967                          { }
8968                      }
8969                    \c_math_toggle_token
8970                    \cr
8971                    \noalign { \skip_vertical:n { 3 pt } \nointerlineskip }
8972                    \@@_math_toggle: #1 \@@_math_toggle: \cr
8973                  }
8974              }
8975            \group_end:
8976          }
8977          { }
8978          { }
8979      }
```

36   The command TikzEveryCell

```
8980  \bool_new:N \l_@@_not_empty_bool
8981  \bool_new:N \l_@@_empty_bool
8982
8983  \keys_define:nn { NiceMatrix / TikzEveryCell }
8984    {
8985      not-empty .code:n =
8986        \bool_lazy_or:nnTF
8987          \l_@@_in_code_after_bool
8988          \g_@@_recreate_cell_nodes_bool
8989          { \bool_set_true:N \l_@@_not_empty_bool }
8990          { \@@_error:n { detection~of~empty~cells } } ,
8991      not-empty .value_forbidden:n = true ,
8992      empty .code:n =
8993        \bool_lazy_or:nnTF
8994          \l_@@_in_code_after_bool
8995          \g_@@_recreate_cell_nodes_bool
8996          { \bool_set_true:N \l_@@_empty_bool }
8997          { \@@_error:n { detection~of~empty~cells } } ,
8998      empty .value_forbidden:n = true ,
8999      unknown .code:n = \@@_error:n { Unknown~key~for~TikzEveryCell }
9000    }
9001
9002
9003  \NewDocumentCommand { \@@_TikzEveryCell } { O { } m }
9004    {
9005      \IfPackageLoadedTF { tikz }
```

```
9006            {
9007              \group_begin:
9008              \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.

```
9009              \tl_set:Nn \l_tmpa_tl { { #2 } }
9010              \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
9011                { \@@_for_a_block:nnnnn ##1 }
9012              \@@_all_the_cells:
9013              \group_end:
9014            }
9015            { \@@_error:n { TikzEveryCell~without~tikz } }
9016        }
9017
9018  \tl_new:N \@@_i_tl
9019  \tl_new:N \@@_j_tl
9020
9021  \cs_new_protected:Nn \@@_all_the_cells:
9022    {
9023      \int_step_variable:nNn { \int_use:c { c@iRow } } \@@_i_tl
9024        {
9025          \int_step_variable:nNn { \int_use:c { c@jCol } } \@@_j_tl
9026            {
9027              \cs_if_exist:cF { cell - \@@_i_tl - \@@_j_tl }
9028                {
9029                  \exp_args:NNe \seq_if_in:NnF \l_@@_corners_cells_seq
9030                    { \@@_i_tl - \@@_j_tl }
9031                    {
9032                      \bool_set_false:N \l_tmpa_bool
9033                      \cs_if_exist:cTF
9034                        { pgf @ sh @ ns @ \@@_env: - \@@_i_tl - \@@_j_tl }
9035                        {
9036                          \bool_if:NF \l_@@_empty_bool
9037                            { \bool_set_true:N \l_tmpa_bool }
9038                        }
9039                        {
9040                          \bool_if:NF \l_@@_not_empty_bool
9041                            { \bool_set_true:N \l_tmpa_bool }
9042                        }
9043                      \bool_if:NT \l_tmpa_bool
9044                        {
9045                          \@@_block_tikz:nnnnV
9046                          \@@_i_tl \@@_j_tl \@@_i_tl \@@_j_tl \l_tmpa_tl
9047                        }
9048                    }
9049                }
9050            }
9051        }
9052    }
9053
9054  \cs_new_protected:Nn \@@_for_a_block:nnnnn
9055    {
9056      \bool_if:NF \l_@@_empty_bool
9057        {
9058          \@@_block_tikz:nnnnV
9059            { #1 } { #2 } { #3 } { #4 } \l_tmpa_tl
9060        }
9061      \@@_mark_cells_of_block:nnnn { #1 } { #2 } { #3 } { #4 }
9062    }
9063
9064  \cs_new_protected:Nn \@@_mark_cells_of_block:nnnn
9065    {
9066      \int_step_inline:nnn { #1 } { #3 }
```

```
9067        {
9068          \int_step_inline:nnn { #2 } { #4 }
9069            { \cs_set:cpn { cell - ##1 - ####1 } { } } }
9070        }
9071    }
```

37   The command \ShowCellNames

```
9072  \NewDocumentCommand \@@_ShowCellNames_CodeBefore { }
9073    {
9074      \dim_gzero_new:N \g_@@_tmpc_dim
9075      \dim_gzero_new:N \g_@@_tmpd_dim
9076      \dim_gzero_new:N \g_@@_tmpe_dim
9077      \int_step_inline:nn \c@iRow
9078        {
9079          \begin { pgfpicture }
9080          \@@_qpoint:n { row - ##1 }
9081          \dim_set_eq:NN \l_tmpa_dim \pgf@y
9082          \@@_qpoint:n { row - \int_eval:n { ##1 + 1 } }
9083          \dim_gset:Nn \g_tmpa_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
9084          \dim_gset:Nn \g_tmpb_dim { \l_tmpa_dim - \pgf@y }
9085          \bool_if:NTF \l_@@_in_code_after_bool
9086          \end { pgfpicture }
9087          \int_step_inline:nn \c@jCol
9088            {
9089              \hbox_set:Nn \l_tmpa_box
9090                { \normalfont \Large \color { red ! 50 } ##1 - ####1 }
9091              \begin { pgfpicture }
9092              \@@_qpoint:n { col - ####1 }
9093              \dim_gset_eq:NN \g_@@_tmpc_dim \pgf@x
9094              \@@_qpoint:n { col - \int_eval:n { ####1 + 1 } }
9095              \dim_gset:Nn \g_@@_tmpd_dim { \pgf@x - \g_@@_tmpc_dim }
9096              \dim_gset_eq:NN \g_@@_tmpe_dim \pgf@x
9097              \endpgfpicture
9098              \end { pgfpicture }
9099              \fp_set:Nn \l_tmpa_fp
9100                {
9101                  \fp_min:nn
9102                    {
9103                      \fp_min:nn
9104                        {
9105                          \dim_ratio:nn
9106                            { \g_@@_tmpd_dim }
9107                            { \box_wd:N \l_tmpa_box }
9108                        }
9109                        {
9110                          \dim_ratio:nn
9111                            { \g_tmpb_dim }
9112                            { \box_ht_plus_dp:N \l_tmpa_box }
9113                        }
9114                    }
9115                    { 1.0 }
9116                }
9117              \box_scale:Nnn \l_tmpa_box
9118                { \fp_use:N \l_tmpa_fp }
9119                { \fp_use:N \l_tmpa_fp }
9120              \pgfpicture
9121              \pgfrememberpicturepositiononpagetrue
9122              \pgf@relevantforpicturesizefalse
9123              \pgftransformshift
9124                {
9125                  \pgfpoint
```

```
9126                   { 0.5 * ( \g_@@_tmpc_dim + \g_@@_tmpe_dim ) }
9127                   { \dim_use:N \g_tmpa_dim }
9128               }
9129             \pgfnode
9130               { rectangle }
9131               { center }
9132               { \box_use:N \l_tmpa_box }
9133               { }
9134               { }
9135             \endpgfpicture
9136           }
9137       }
9138   }
9139 \NewDocumentCommand \@@_ShowCellNames { }
9140   {
9141     \bool_if:NT \l_@@_in_code_after_bool
9142       {
9143         \pgfpicture
9144         \pgfrememberpicturepositiononpagetrue
9145         \pgf@relevantforpicturesizefalse
9146         \pgfpathrectanglecorners
9147           { \@@_qpoint:n { 1 } }
9148           {
9149             \@@_qpoint:n
9150               { \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
9151           }
9152         \pgfsetfillopacity { 0.75 }
9153         \pgfsetfillcolor { white }
9154         \pgfusepathqfill
9155         \endpgfpicture
9156       }
9157     \dim_gzero_new:N \g_@@_tmpc_dim
9158     \dim_gzero_new:N \g_@@_tmpd_dim
9159     \dim_gzero_new:N \g_@@_tmpe_dim
9160     \int_step_inline:nn \c@iRow
9161       {
9162         \bool_if:NTF \l_@@_in_code_after_bool
9163           {
9164             \pgfpicture
9165             \pgfrememberpicturepositiononpagetrue
9166             \pgf@relevantforpicturesizefalse
9167           }
9168           { \begin { pgfpicture } }
9169         \@@_qpoint:n { row - ##1 }
9170         \dim_set_eq:NN \l_tmpa_dim \pgf@y
9171         \@@_qpoint:n { row - \int_eval:n { ##1 + 1 } }
9172         \dim_gset:Nn \g_tmpa_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
9173         \dim_gset:Nn \g_tmpb_dim { \l_tmpa_dim - \pgf@y }
9174         \bool_if:NTF \l_@@_in_code_after_bool
9175           { \endpgfpicture }
9176           { \end { pgfpicture } }
9177         \int_step_inline:nn \c@jCol
9178           {
9179             \hbox_set:Nn \l_tmpa_box
9180               {
9181                 \normalfont \Large \sffamily \bfseries
9182                 \bool_if:NTF \l_@@_in_code_after_bool
9183                   { \color { red } }
9184                   { \color { red ! 50 } }
9185                 ##1 - ####1
9186               }
9187             \bool_if:NTF \l_@@_in_code_after_bool
9188               {
```

206

```
9189            \pgfpicture
9190            \pgfrememberpicturepositiononpagetrue
9191            \pgf@relevantforpicturesizefalse
9192          }
9193          { \begin { pgfpicture } }
9194        \@@_qpoint:n { col - ####1 }
9195        \dim_gset_eq:NN \g_@@_tmpc_dim \pgf@x
9196        \@@_qpoint:n { col - \int_eval:n { ####1 + 1 } }
9197        \dim_gset:Nn \g_@@_tmpd_dim { \pgf@x - \g_@@_tmpc_dim }
9198        \dim_gset_eq:NN \g_@@_tmpe_dim \pgf@x
9199        \bool_if:NTF \l_@@_in_code_after_bool
9200          { \endpgfpicture }
9201          { \end { pgfpicture } }
9202        \fp_set:Nn \l_tmpa_fp
9203          {
9204            \fp_min:nn
9205              {
9206                \fp_min:nn
9207                  { \dim_ratio:nn \g_@@_tmpd_dim { \box_wd:N \l_tmpa_box } }
9208                  { \dim_ratio:nn \g_tmpb_dim { \box_ht_plus_dp:N \l_tmpa_box } }
9209              }
9210              { 1.0 }
9211          }
9212        \box_scale:Nnn \l_tmpa_box { \fp_use:N \l_tmpa_fp } { \fp_use:N \l_tmpa_fp }
9213        \pgfpicture
9214        \pgfrememberpicturepositiononpagetrue
9215        \pgf@relevantforpicturesizefalse
9216        \pgftransformshift
9217          {
9218            \pgfpoint
9219              { 0.5 * ( \g_@@_tmpc_dim + \g_@@_tmpe_dim ) }
9220              { \dim_use:N \g_tmpa_dim }
9221          }
9222        \pgfnode
9223          { rectangle }
9224          { center }
9225          { \box_use:N \l_tmpa_box }
9226          { }
9227          { }
9228        \endpgfpicture
9229        }
9230      }
9231  }
```

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.

```
9232 \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.

```
9233 \bool_new:N \g_@@_footnote_bool
9234 \msg_new:nnnn { nicematrix } { Unknown~key~for~package }
9235   {
9236     The~key~'\l_keys_key_str'~is~unknown. \\
```

```
9237      That~key~will~be~ignored. \\
9238      For~a~list~of~the~available~keys,~type~H~<return>.
9239    }
9240    {
9241      The~available~keys~are~(in~alphabetic~order):~
9242      footnote,~
9243      footnotehyper,~
9244      messages-for-Overleaf,~
9245      no-test-for-array,~
9246      renew-dots,~and~
9247      renew-matrix.
9248    }
9249  \keys_define:nn { NiceMatrix / Package }
9250    {
9251      renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
9252      renew-dots .value_forbidden:n = true ,
9253      renew-matrix .code:n = \@@_renew_matrix: ,
9254      renew-matrix .value_forbidden:n = true ,
9255      messages-for-Overleaf .bool_set:N = \g_@@_messages_for_Overleaf_bool ,
9256      footnote .bool_set:N = \g_@@_footnote_bool ,
9257      footnotehyper .bool_set:N = \g_@@_footnotehyper_bool ,
9258      no-test-for-array .bool_set:N = \g_@@_no_test_for_array_bool ,
9259      no-test-for-array .default:n = true ,
9260      unknown .code:n = \@@_error:n { Unknown~key~for~package }
9261    }
9262  \ProcessKeysOptions { NiceMatrix / Package }


9263  \@@_msg_new:nn { footnote~with~footnotehyper~package }
9264    {
9265      You~can't~use~the~option~'footnote'~because~the~package~
9266      footnotehyper~has~already~been~loaded.~
9267      If~you~want,~you~can~use~the~option~'footnotehyper'~and~the~footnotes~
9268      within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
9269      of~the~package~footnotehyper.\\
9270      The~package~footnote~won't~be~loaded.
9271    }
9272  \@@_msg_new:nn { footnotehyper~with~footnote~package }
9273    {
9274      You~can't~use~the~option~'footnotehyper'~because~the~package~
9275      footnote~has~already~been~loaded.~
9276      If~you~want,~you~can~use~the~option~'footnote'~and~the~footnotes~
9277      within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
9278      of~the~package~footnote.\\
9279      The~package~footnotehyper~won't~be~loaded.
9280    }


9281  \bool_if:NT \g_@@_footnote_bool
9282    {
```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```
9283      \IfClassLoadedTF { beamer }
9284        { \bool_set_false:N \g_@@_footnote_bool }
9285        {
9286          \IfPackageLoadedTF { footnotehyper }
9287            { \@@_error:n { footnote~with~footnotehyper~package } }
9288            { \usepackage { footnote } }
9289        }
9290    }
9291  \bool_if:NT \g_@@_footnotehyper_bool
9292    {
```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```
9293       \IfClassLoadedTF { beamer }
9294         { \bool_set_false:N \g_@@_footnote_bool }
9295         {
9296           \IfPackageLoadedTF { footnote }
9297             { \@@_error:n { footnotehyper~with~footnote~package } }
9298             { \usepackage { footnotehyper } }
9299         }
9300       \bool_set_true:N \g_@@_footnote_bool
9301   }
```

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.

```
9302 \bool_new:N \l_@@_underscore_loaded_bool
9303 \IfPackageLoadedTF { underscore }
9304   { \bool_set_true:N \l_@@_underscore_loaded_bool }
9305   { }
9306 \hook_gput_code:nnn { begindocument } { . }
9307   {
9308     \bool_if:NF \l_@@_underscore_loaded_bool
9309       {
9310         \IfPackageLoadedTF { underscore }
9311           { \@@_error:n { underscore~after~nicematrix } }
9312           { }
9313       }
9314   }
```

40   Error messages of the package

```
9315 \bool_if:NTF \g_@@_messages_for_Overleaf_bool
9316   { \str_const:Nn \c_@@_available_keys_str { } }
9317   {
9318     \str_const:Nn \c_@@_available_keys_str
9319       { For~a~list~of~the~available~keys,~type~H~<return>. }
9320   }
9321 \seq_new:N \g_@@_types_of_matrix_seq
9322 \seq_gset_from_clist:Nn \g_@@_types_of_matrix_seq
9323   {
9324     NiceMatrix ,
9325     pNiceMatrix , bNiceMatrix , vNiceMatrix, BNiceMatrix, VNiceMatrix
9326   }
9327 \seq_gset_map_x:NNn \g_@@_types_of_matrix_seq \g_@@_types_of_matrix_seq
9328   { \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`.

```
9329 \cs_new_protected:Npn \@@_error_too_much_cols:
9330   {
9331     \seq_if_in:NoTF \g_@@_types_of_matrix_seq \g_@@_name_env_str
9332       {
9333         \int_compare:nNnTF \l_@@_last_col_int = { -2 }
9334           { \@@_fatal:n { too~much~cols~for~matrix } }
9335           {
9336             \int_compare:nNnTF \l_@@_last_col_int = { -1 }
9337               { \@@_fatal:n { too~much~cols~for~matrix } }
9338               {
9339                 \bool_if:NF \l_@@_last_col_without_value_bool
9340                   { \@@_fatal:n { too~much~cols~for~matrix~with~last~col } }
9341               }
9342           }
9343       }
9344       { \@@_fatal:nn { too~much~cols~for~array } }
9345   }
```

The following command must *not* be protected since it's used in an error message.

```
9346 \cs_new:Npn \@@_message_hdotsfor:
9347   {
9348     \tl_if_empty:oF \g_@@_HVdotsfor_lines_tl
9349       { ~Maybe~your~use~of~\token_to_str:N \Hdotsfor\ is~incorrect.}
9350   }
9351 \@@_msg_new:nn { hvlines,~rounded-corners~and~corners }
9352   {
9353     Incompatible~options.\\
9354     You~should~not~use~'hvlines',~'rounded-corners'~and~'corners'~at~this~time.\\
9355     The~output~will~not~be~reliable.
9356   }
9357 \@@_msg_new:nn { negative~weight }
9358   {
9359     Negative~weight.\\
9360     The~weight~of~the~'X'~columns~must~be~positive~and~you~have~used~
9361     the~value~'\int_use:N \l_@@_weight_int'.\\
9362     The~absolute~value~will~be~used.
9363   }
9364 \@@_msg_new:nn { last~col~not~used }
9365   {
9366     Column~not~used.\\
9367     The~key~'last-col'~is~in~force~but~you~have~not~used~that~last~column~
9368     in~your~\@@_full_name_env:.~However,~you~can~go~on.
9369   }
9370 \@@_msg_new:nn { too~much~cols~for~matrix~with~last~col }
9371   {
9372     Too~much~columns.\\
9373     In~the~row~\int_eval:n { \c@iRow },~
9374     you~try~to~use~more~columns~
9375     than~allowed~by~your~\@@_full_name_env:.\@@_message_hdotsfor:\
9376     The~maximal~number~of~columns~is~\int_eval:n { \l_@@_last_col_int - 1 }~
9377     (plus~the~exterior~columns).~This~error~is~fatal.
9378   }
9379 \@@_msg_new:nn { too~much~cols~for~matrix }
9380   {
9381     Too~much~columns.\\
9382     In~the~row~\int_eval:n { \c@iRow },~
9383     you~try~to~use~more~columns~than~allowed~by~your~
9384     \@@_full_name_env:.\@@_message_hdotsfor:\ Recall~that~the~maximal~
9385     number~of~columns~for~a~matrix~(excepted~the~potential~exterior~
9386     columns)~is~fixed~by~the~LaTeX~counter~'MaxMatrixCols'.~
```

```
9387        Its~current~value~is~\int_use:N \c@MaxMatrixCols\ (use~
9388        \token_to_str:N \setcounter\ to~change~that~value).~
9389        This~error~is~fatal.
9390      }

9391  \@@_msg_new:nn { too~much~cols~for~array }
9392    {
9393      Too~much~columns.\\
9394      In~the~row~\int_eval:n { \c@iRow },~
9395      ~you~try~to~use~more~columns~than~allowed~by~your~
9396      \@@_full_name_env:.\@@_message_hdotsfor:\ The~maximal~number~of~columns~is~
9397      \int_use:N \g_@@_static_num_of_col_int\
9398      ~(plus~the~potential~exterior~ones).
9399      This~error~is~fatal.
9400    }
9401  \@@_msg_new:nn { columns~not~used }
9402    {
9403      Columns~not~used.\\
9404      The~preamble~of~your~\@@_full_name_env:\ announces~\int_use:N
9405      \g_@@_static_num_of_col_int\ columns~but~you~use~only~\int_use:N \c@jCol.\\
9406      The~columns~you~did~not~used~won't~be~created.\\
9407      You~won't~have~similar~error~message~till~the~end~of~the~document.
9408    }
9409  \@@_msg_new:nn { empty~preamble }
9410    {
9411      Empty~preamble.\\
9412      The~preamble~of~your~\@@_full_name_env:\ is~empty.\\
9413      This~error~is~fatal.
9414    }
9415  \@@_msg_new:nn { in~first~col }
9416    {
9417      Erroneous~use.\\
9418      You~can't~use~the~command~#1 in~the~first~column~(number~0)~of~the~array.\\
9419      That~command~will~be~ignored.
9420    }
9421  \@@_msg_new:nn { in~last~col }
9422    {
9423      Erroneous~use.\\
9424      You~can't~use~the~command~#1 in~the~last~column~(exterior)~of~the~array.\\
9425      That~command~will~be~ignored.
9426    }
9427  \@@_msg_new:nn { in~first~row }
9428    {
9429      Erroneous~use.\\
9430      You~can't~use~the~command~#1 in~the~first~row~(number~0)~of~the~array.\\
9431      That~command~will~be~ignored.
9432    }
9433  \@@_msg_new:nn { in~last~row }
9434    {
9435      You~can't~use~the~command~#1 in~the~last~row~(exterior)~of~the~array.\\
9436      That~command~will~be~ignored.
9437    }
9438  \@@_msg_new:nn { caption~outside~float }
9439    {
9440      Key~caption~forbidden.\\
9441      You~can't~use~the~key~'caption'~because~you~are~not~in~a~floating~
9442      environment.~This~key~will~be~ignored.
9443    }
9444  \@@_msg_new:nn { short-caption~without~caption }
9445    {
```

211

```
9446        You~should~not~use~the~key~'short-caption'~without~'caption'.~
9447        However,~your~'short-caption'~will~be~used~as~'caption'.
9448      }
9449    \@@_msg_new:nn { double~closing~delimiter }
9450      {
9451        Double~delimiter.\\
9452        You~can't~put~a~second~closing~delimiter~"#1"~just~after~a~first~closing~
9453        delimiter.~This~delimiter~will~be~ignored.
9454      }
9455    \@@_msg_new:nn { delimiter~after~opening }
9456      {
9457        Double~delimiter.\\
9458        You~can't~put~a~second~delimiter~"#1"~just~after~a~first~opening~
9459        delimiter.~That~delimiter~will~be~ignored.
9460      }
9461    \@@_msg_new:nn { bad~option~for~line-style }
9462      {
9463        Bad~line~style.\\
9464        Since~you~haven't~loaded~Tikz,~the~only~value~you~can~give~to~'line-style'~
9465        is~'standard'.~That~key~will~be~ignored.
9466      }
9467    \@@_msg_new:nn { Identical~notes~in~caption }
9468      {
9469        Identical~tabular~notes.\\
9470        You~can't~put~several~notes~with~the~same~content~in~
9471        \token_to_str:N \caption\ (but~you~can~in~the~main~tabular).\\
9472        If~you~go~on,~the~output~will~probably~be~erroneous.
9473      }
9474    \@@_msg_new:nn { tabularnote~below~the~tabular }
9475      {
9476        \token_to_str:N \tabularnote\ forbidden\\
9477        You~can't~use~\token_to_str:N \tabularnote\ in~the~caption~
9478        of~your~tabular~because~the~caption~will~be~composed~below~
9479        the~tabular.~If~you~want~the~caption~above~the~tabular~use~the~
9480        key~'caption-above'~in~\token_to_str:N \NiceMatrixOptions.\\
9481        Your~\token_to_str:N \tabularnote\ will~be~discarded~and~
9482        no~similar~error~will~raised~in~this~document.
9483      }
9484    \@@_msg_new:nn { Unknown~key~for~rules }
9485      {
9486        Unknown~key.\\
9487        There~is~only~two~keys~available~here:~width~and~color.\\
9488        Your~key~'\l_keys_key_str'~will~be~ignored.
9489      }
9490    \@@_msg_new:nn { Unknown~key~for~TikzEveryCell }
9491      {
9492        Unknown~key.\\
9493        There~is~only~two~keys~available~here:~
9494        'empty'~and~'not-empty'.\\
9495        Your~key~'\l_keys_key_str'~will~be~ignored.
9496      }
9497    \@@_msg_new:nn { Unknown~key~for~rotate }
9498      {
9499        Unknown~key.\\
9500        The~only~key~available~here~is~'c'.\\
9501        Your~key~'\l_keys_key_str'~will~be~ignored.
9502      }
9503    \@@_msg_new:nnn { Unknown~key~for~custom-line }
9504      {
9505        Unknown~key.\\
```

```
9506        The~key~'\l_keys_key_str'~is~unknown~in~a~'custom-line'.~
9507        It~you~go~on,~you~will~probably~have~other~errors. \\
9508        \c_@@_available_keys_str
9509      }
9510      {
9511        The~available~keys~are~(in~alphabetic~order):~
9512        ccommand,~
9513        color,~
9514        command,~
9515        dotted,~
9516        letter,~
9517        multiplicity,~
9518        sep-color,~
9519        tikz,~and~total-width.
9520      }
9521    \@@_msg_new:nnn { Unknown~key~for~xdots }
9522      {
9523        Unknown~key.\\
9524        The~key~'\l_keys_key_str'~is~unknown~for~a~command~for~drawing~dotted~rules.\\
9525        \c_@@_available_keys_str
9526      }
9527      {
9528        The~available~keys~are~(in~alphabetic~order):~
9529        'color',~
9530        'horizontal-labels',~
9531        'inter',~
9532        'line-style',~
9533        'radius',~
9534        'shorten',~
9535        'shorten-end'~and~'shorten-start'.
9536      }
9537    \@@_msg_new:nn { Unknown~key~for~rowcolors }
9538      {
9539        Unknown~key.\\
9540        As~for~now,~there~is~only~two~keys~available~here:~'cols'~and~'respect-blocks'~
9541        (and~you~try~to~use~'\l_keys_key_str')\\
9542        That~key~will~be~ignored.
9543      }
9544    \@@_msg_new:nn { label~without~caption }
9545      {
9546        You~can't~use~the~key~'label'~in~your~'{NiceTabular}'~because~
9547        you~have~not~used~the~key~'caption'.~The~key~'label'~will~be~ignored.
9548      }
9549    \@@_msg_new:nn { W~warning }
9550      {
9551        Line~\msg_line_number:.~The~cell~is~too~wide~for~your~column~'W'~
9552        (row~\int_use:N \c@iRow).
9553      }
9554    \@@_msg_new:nn { Construct~too~large }
9555      {
9556        Construct~too~large.\\
9557        Your~command~\token_to_str:N #1
9558        can't~be~drawn~because~your~matrix~is~too~small.\\
9559        That~command~will~be~ignored.
9560      }
9561    \@@_msg_new:nn { underscore~after~nicematrix }
9562      {
9563        Problem~with~'underscore'.\\
9564        The~package~'underscore'~should~be~loaded~before~'nicematrix'.~
9565        You~can~go~on~but~you~won't~be~able~to~write~something~such~as:\\
9566        '\token_to_str:N \Cdots\token_to_str:N _{n~\token_to_str:N \text{~times}}'.
```

```
9567      }
9568  \@@_msg_new:nn { ampersand~in~light-syntax }
9569      {
9570        Ampersand~forbidden.\\
9571        You~can't~use~an~ampersand~(\token_to_str:N &)~to~separate~columns~because~
9572        ~the~key~'light-syntax'~is~in~force.~This~error~is~fatal.
9573      }
9574  \@@_msg_new:nn { double-backslash~in~light-syntax }
9575      {
9576        Double~backslash~forbidden.\\
9577        You~can't~use~\token_to_str:N
9578        \\~to~separate~rows~because~the~key~'light-syntax'~
9579        is~in~force.~You~must~use~the~character~'\l_@@_end_of_row_tl'~
9580        (set~by~the~key~'end-of-row').~This~error~is~fatal.
9581      }
9582  \@@_msg_new:nn { hlines~with~color }
9583      {
9584        Incompatible~keys.\\
9585        You~can't~use~the~keys~'hlines',~'vlines'~or~'hvlines'~for~a~
9586        '\token_to_str:N \Block'~when~the~key~'color'~or~'draw'~is~used.\\
9587        However,~you~can~put~several~commands~\token_to_str:N \Block.\\
9588        Your~key~will~be~discarded.
9589      }
9590  \@@_msg_new:nn { bad~value~for~baseline }
9591      {
9592        Bad~value~for~baseline.\\
9593        The~value~given~to~'baseline'~(\int_use:N \l_tmpa_int)~is~not~
9594        valid.~The~value~must~be~between~\int_use:N \l_@@_first_row_int\ and~
9595        \int_use:N \g_@@_row_total_int\ or~equal~to~'t',~'c'~or~'b'~or~of~
9596        the~form~'line-i'.\\
9597        A~value~of~1~will~be~used.
9598      }
9599  \@@_msg_new:nn { detection~of~empty~cells }
9600      {
9601        Problem~with~'not-empty'\\
9602        For~technical~reasons,~you~must~activate~
9603        'create-cell-nodes'~in~\token_to_str:N \CodeBefore\
9604        in~order~to~use~the~key~'\l_keys_key_str'.\\
9605        That~key~will~be~ignored.
9606      }
9607  \@@_msg_new:nn { siunitx~not~loaded }
9608      {
9609        siunitx~not~loaded\\
9610        You~can't~use~the~columns~'S'~because~'siunitx'~is~not~loaded.\\
9611        That~error~is~fatal.
9612      }
9613  \@@_msg_new:nn { ragged2e~not~loaded }
9614      {
9615        You~have~to~load~'ragged2e'~in~order~to~use~the~key~'\l_keys_key_str'~in~
9616        your~column~'\l_@@_vpos_col_str'~(or~'X').~The~key~'\str_lowercase:o
9617        \l_keys_key_str'~will~be~used~instead.
9618      }
9619  \@@_msg_new:nn { Invalid~name }
9620      {
9621        Invalid~name.\\
9622        You~can't~give~the~name~'\l_keys_value_tl'~to~a~\token_to_str:N
9623        \SubMatrix\ of~your~\@@_full_name_env:.\\
9624        A~name~must~be~accepted~by~the~regular~expression~[A-Za-z][A-Za-z0-9]*.\\
9625        This~key~will~be~ignored.
9626      }
```

```
9627  \@@_msg_new:nn { Wrong~line~in~SubMatrix }
9628    {
9629      Wrong~line.\\
9630      You~try~to~draw~a~#1~line~of~number~'#2'~in~a~
9631      \token_to_str:N \SubMatrix\ of~your~\@@_full_name_env:\ but~that~
9632      number~is~not~valid.~It~will~be~ignored.
9633    }
9634  \@@_msg_new:nn { Impossible~delimiter }
9635    {
9636      Impossible~delimiter.\\
9637      It's~impossible~to~draw~the~#1~delimiter~of~your~
9638      \token_to_str:N \SubMatrix\ because~all~the~cells~are~empty~
9639      in~that~column.
9640      \bool_if:NT \l_@@_submatrix_slim_bool
9641        { ~Maybe~you~should~try~without~the~key~'slim'. } \\
9642      This~\token_to_str:N \SubMatrix\ will~be~ignored.
9643    }
9644  \@@_msg_new:nnn { width~without~X~columns }
9645    {
9646      You~have~used~the~key~'width'~but~you~have~put~no~'X'~column.~
9647      That~key~will~be~ignored.
9648    }
9649    {
9650      This~message~is~the~message~'width~without~X~columns'~
9651      of~the~module~'nicematrix'.~
9652      The~experimented~users~can~disable~that~message~with~
9653      \token_to_str:N \msg_redirect_name:nnn.\\
9654    }

9656  \@@_msg_new:nn { key~multiplicity~with~dotted }
9657    {
9658      Incompatible~keys. \\
9659      You~have~used~the~key~'multiplicity'~with~the~key~'dotted'~
9660      in~a~'custom-line'.~They~are~incompatible. \\
9661      The~key~'multiplicity'~will~be~discarded.
9662    }
9663  \@@_msg_new:nn { empty~environment }
9664    {
9665      Empty~environment.\\
9666      Your~\@@_full_name_env:\ is~empty.~This~error~is~fatal.
9667    }
9668  \@@_msg_new:nn { No~letter~and~no~command }
9669    {
9670      Erroneous~use.\\
9671      Your~use~of~'custom-line'~is~no-op~since~you~don't~have~used~the~
9672      key~'letter'~(for~a~letter~for~vertical~rules)~nor~the~keys~'command'~or~
9673      ~'ccommand'~(to~draw~horizontal~rules).\\
9674      However,~you~can~go~on.
9675    }
9676  \@@_msg_new:nn { Forbidden~letter }
9677    {
9678      Forbidden~letter.\\
9679      You~can't~use~the~letter~'#1'~for~a~customized~line.\\
9680      It~will~be~ignored.
9681    }
9682  \@@_msg_new:nn { Several~letters }
9683    {
9684      Wrong~name.\\
9685      You~must~use~only~one~letter~as~value~for~the~key~'letter'~(and~you~
9686      have~used~'\l_@@_letter_str').\\
9687      It~will~be~ignored.
```

215

```
9688        }
9689    \@@_msg_new:nn { Delimiter~with~small }
9690      {
9691        Delimiter~forbidden.\\
9692        You~can't~put~a~delimiter~in~the~preamble~of~your~\@@_full_name_env:\
9693        because~the~key~'small'~is~in~force.\\
9694        This~error~is~fatal.
9695      }
9696    \@@_msg_new:nn { unknown~cell~for~line~in~CodeAfter }
9697      {
9698        Unknown~cell.\\
9699        Your~command~\token_to_str:N\line\{#1\}\{#2\}~in~
9700        the~\token_to_str:N \CodeAfter\ of~your~\@@_full_name_env:\
9701        can't~be~executed~because~a~cell~doesn't~exist.\\
9702        This~command~\token_to_str:N \line\ will~be~ignored.
9703      }
9704    \@@_msg_new:nnn { Duplicate~name~for~SubMatrix }
9705      {
9706        Duplicate~name.\\
9707        The~name~'#1'~is~already~used~for~a~\token_to_str:N \SubMatrix\
9708        in~this~\@@_full_name_env:.\\
9709        This~key~will~be~ignored.\\
9710        \bool_if:NF \g_@@_messages_for_Overleaf_bool
9711          { For~a~list~of~the~names~already~used,~type~H~<return>. }
9712      }
9713      {
9714        The~names~already~defined~in~this~\@@_full_name_env:\ are:~
9715        \seq_use:Nnnn \g_@@_submatrix_names_seq { ~and~ } { ,~ } { ~and~ }.
9716      }
9717    \@@_msg_new:nn { r~or~l~with~preamble }
9718      {
9719        Erroneous~use.\\
9720        You~can't~use~the~key~'\l_keys_key_str'~in~your~\@@_full_name_env:.~
9721        You~must~specify~the~alignment~of~your~columns~with~the~preamble~of~
9722        your~\@@_full_name_env:.\\
9723        This~key~will~be~ignored.
9724      }
9725    \@@_msg_new:nn { Hdotsfor~in~col~0 }
9726      {
9727        Erroneous~use.\\
9728        You~can't~use~\token_to_str:N \Hdotsfor\ in~an~exterior~column~of~
9729        the~array.~This~error~is~fatal.
9730      }
9731    \@@_msg_new:nn { bad~corner }
9732      {
9733        Bad~corner.\\
9734        #1~is~an~incorrect~specification~for~a~corner~(in~the~key~
9735        'corners').~The~available~values~are:~NW,~SW,~NE~and~SE.\\
9736        This~specification~of~corner~will~be~ignored.
9737      }
9738    \@@_msg_new:nn { bad~border }
9739      {
9740        Bad~border.\\
9741        \l_keys_key_str\space~is~an~incorrect~specification~for~a~border~
9742        (in~the~key~'borders'~of~the~command~\token_to_str:N \Block).~
9743        The~available~values~are:~left,~right,~top~and~bottom~(and~you~can~
9744        also~use~the~key~'tikz'
9745        \IfPackageLoadedTF { tikz }
9746          { }
9747          {~if~you~load~the~LaTeX~package~'tikz'}).\\
9748        This~specification~of~border~will~be~ignored.
```

216

```
9749    }
9750  \@@_msg_new:nn { TikzEveryCell~without~tikz }
9751    {
9752      TikZ~not~loaded.\\
9753      You~can't~use~\token_to_str:N \TikzEveryCell\
9754      because~you~have~not~loaded~tikz.~
9755      This~command~will~be~ignored.
9756    }
9757  \@@_msg_new:nn { tikz~key~without~tikz }
9758    {
9759      TikZ~not~loaded.\\
9760      You~can't~use~the~key~'tikz'~for~the~command~'\token_to_str:N
9761      \Block'~because~you~have~not~loaded~tikz.~
9762      This~key~will~be~ignored.
9763    }
9764  \@@_msg_new:nn { last-col~non~empty~for~NiceArray }
9765    {
9766      Erroneous~use.\\
9767      In~the~\@@_full_name_env:,~you~must~use~the~key~
9768      'last-col'~without~value.\\
9769      However,~you~can~go~on~for~this~time~
9770      (the~value~'\l_keys_value_tl'~will~be~ignored).
9771    }
9772  \@@_msg_new:nn { last-col~non~empty~for~NiceMatrixOptions }
9773    {
9774      Erroneous~use.\\
9775      In~\token_to_str:N \NiceMatrixOptions,~you~must~use~the~key~
9776      'last-col'~without~value.\\
9777      However,~you~can~go~on~for~this~time~
9778      (the~value~'\l_keys_value_tl'~will~be~ignored).
9779    }
9780  \@@_msg_new:nn { Block~too~large~1 }
9781    {
9782      Block~too~large.\\
9783      You~try~to~draw~a~block~in~the~cell~#1-#2~of~your~matrix~but~the~matrix~is~
9784      too~small~for~that~block. \\
9785      This~block~and~maybe~others~will~be~ignored.
9786    }
9787  \@@_msg_new:nn { Block~too~large~2 }
9788    {
9789      Block~too~large.\\
9790      The~preamble~of~your~\@@_full_name_env:\ announces~\int_use:N
9791      \g_@@_static_num_of_col_int\
9792      columns~but~you~use~only~\int_use:N \c@jCol\ and~that's~why~a~block~
9793      specified~in~the~cell~#1-#2~can't~be~drawn.~You~should~add~some~ampersands~
9794      (&)~at~the~end~of~the~first~row~of~your~\@@_full_name_env:.\\
9795      This~block~and~maybe~others~will~be~ignored.
9796    }
9797  \@@_msg_new:nn { unknown~column~type }
9798    {
9799      Bad~column~type.\\
9800      The~column~type~'#1'~in~your~\@@_full_name_env:\
9801      is~unknown. \\
9802      This~error~is~fatal.
9803    }
9804  \@@_msg_new:nn { unknown~column~type~S }
9805    {
9806      Bad~column~type.\\
9807      The~column~type~'S'~in~your~\@@_full_name_env:\ is~unknown. \\
9808      If~you~want~to~use~the~column~type~'S'~of~siunitx,~you~should~
```

```
9809      load~that~package. \\
9810      This~error~is~fatal.
9811    }
9812  \@@_msg_new:nn { tabularnote~forbidden }
9813    {
9814      Forbidden~command.\\
9815      You~can't~use~the~command~\token_to_str:N\tabularnote\
9816      ~here.~This~command~is~available~only~in~
9817      \{NiceTabular\},~\{NiceTabular*\}~and~\{NiceTabularX\}~or~in~
9818      the~argument~of~a~command~\token_to_str:N \caption\ included~
9819      in~an~environment~{table}. \\
9820      This~command~will~be~ignored.
9821    }
9822  \@@_msg_new:nn { borders~forbidden }
9823    {
9824      Forbidden~key.\\
9825      You~can't~use~the~key~'borders'~of~the~command~\token_to_str:N \Block\
9826      because~the~option~'rounded-corners'~
9827      is~in~force~with~a~non-zero~value.\\
9828      This~key~will~be~ignored.
9829    }
9830  \@@_msg_new:nn { bottomrule~without~booktabs }
9831    {
9832      booktabs~not~loaded.\\
9833      You~can't~use~the~key~'tabular/bottomrule'~because~you~haven't~
9834      loaded~'booktabs'.\\
9835      This~key~will~be~ignored.
9836    }
9837  \@@_msg_new:nn { enumitem~not~loaded }
9838    {
9839      enumitem~not~loaded.\\
9840      You~can't~use~the~command~\token_to_str:N\tabularnote\
9841      ~because~you~haven't~loaded~'enumitem'.\\
9842      All~the~commands~\token_to_str:N\tabularnote\ will~be~
9843      ignored~in~the~document.
9844    }
9845  \@@_msg_new:nn { tikz~without~tikz }
9846    {
9847      Tikz~not~loaded.\\
9848      You~can't~use~the~key~'tikz'~here~because~Tikz~is~not~
9849      loaded.~If~you~go~on,~that~key~will~be~ignored.
9850    }
9851  \@@_msg_new:nn { tikz~in~custom-line~without~tikz }
9852    {
9853      Tikz~not~loaded.\\
9854      You~have~used~the~key~'tikz'~in~the~definition~of~a~
9855      customized~line~(with~'custom-line')~but~tikz~is~not~loaded.~
9856      You~can~go~on~but~you~will~have~another~error~if~you~actually~
9857      use~that~custom~line.
9858    }
9859  \@@_msg_new:nn { tikz~in~borders~without~tikz }
9860    {
9861      Tikz~not~loaded.\\
9862      You~have~used~the~key~'tikz'~in~a~key~'borders'~(of~a~
9863      command~'\token_to_str:N\Block')~but~tikz~is~not~loaded.~
9864      That~key~will~be~ignored.
9865    }
9866  \@@_msg_new:nn { without~color-inside }
9867    {
9868      If~order~to~use~\token_to_str:N \cellcolor,~\token_to_str:N \rowcolor,~
```

```
9869      \token_to_str:N \rowcolors\ or~\token_to_str:N \rowlistcolors\
9870      outside~\token_to_str:N \CodeBefore,~you~
9871      should~have~used~the~key~'color-inside'~in~your~\@@_full_name_env:.\\
9872      You~can~go~on~but~you~may~need~more~compilations.
9873    }
9874  \@@_msg_new:nn { color~in~custom-line~with~tikz }
9875    {
9876      Erroneous~use.\\
9877      In~a~'custom-line',~you~have~used~both~'tikz'~and~'color',~
9878      which~is~forbidden~(you~should~use~'color'~inside~the~key~'tikz').~
9879      The~key~'color'~will~be~discarded.
9880    }
9881  \@@_msg_new:nn { Wrong~last~row }
9882    {
9883      Wrong~number.\\
9884      You~have~used~'last-row=\int_use:N \l_@@_last_row_int'~but~your~
9885      \@@_full_name_env:\ seems~to~have~\int_use:N \c@iRow \ rows.~
9886      If~you~go~on,~the~value~of~\int_use:N \c@iRow \ will~be~used~for~
9887      last~row.~You~can~avoid~this~problem~by~using~'last-row'~
9888      without~value~(more~compilations~might~be~necessary).
9889    }
9890  \@@_msg_new:nn { Yet~in~env }
9891    {
9892      Nested~environments.\\
9893      Environments~of~nicematrix~can't~be~nested.\\
9894      This~error~is~fatal.
9895    }
9896  \@@_msg_new:nn { Outside~math~mode }
9897    {
9898      Outside~math~mode.\\
9899      The~\@@_full_name_env:\ can~be~used~only~in~math~mode~
9900      (and~not~in~\token_to_str:N \vcenter).\\
9901      This~error~is~fatal.
9902    }
9903  \@@_msg_new:nn { One~letter~allowed }
9904    {
9905      Bad~name.\\
9906      The~value~of~key~'\l_keys_key_str'~must~be~of~length~1.\\
9907      It~will~be~ignored.
9908    }
9909  \@@_msg_new:nn { TabularNote~in~CodeAfter }
9910    {
9911      Environment~{TabularNote}~forbidden.\\
9912      You~must~use~{TabularNote}~at~the~end~of~your~{NiceTabular}~
9913      but~*before*~the~\token_to_str:N \CodeAfter.\\
9914      This~environment~{TabularNote}~will~be~ignored.
9915    }
9916  \@@_msg_new:nn { varwidth~not~loaded }
9917    {
9918      varwidth~not~loaded.\\
9919      You~can't~use~the~column~type~'V'~because~'varwidth'~is~not~
9920      loaded.\\
9921      Your~column~will~behave~like~'p'.
9922    }
9923  \@@_msg_new:nnn { Unknow~key~for~RulesBis }
9924    {
9925      Unkown~key.\\
9926      Your~key~'\l_keys_key_str'~is~unknown~for~a~rule.\\
9927      \c_@@_available_keys_str
9928    }
```

```
9929    {
9930      The~available~keys~are~(in~alphabetic~order):~
9931      color,~
9932      dotted,~
9933      multiplicity,~
9934      sep-color,~
9935      tikz,~and~total-width.
9936    }
9937
9938  \@@_msg_new:nnn { Unknown~key~for~Block }
9939    {
9940      Unknown~key.\\
9941      The~key~'\l_keys_key_str'~is~unknown~for~the~command~\token_to_str:N
9942      \Block.\\ It~will~be~ignored. \\
9943      \c_@@_available_keys_str
9944    }
9945    {
9946      The~available~keys~are~(in~alphabetic~order):~b,~B,~borders,~c,~draw,~fill,~
9947      hlines,~hvlines,~l,~line-width,~name,~opacity,~rounded-corners,~r,~
9948      respect-arraystretch,~t,~T,~tikz,~transparent~and~vlines.
9949    }
9950  \@@_msg_new:nnn { Unknown~key~for~Brace }
9951    {
9952      Unknown~key.\\
9953      The~key~'\l_keys_key_str'~is~unknown~for~the~commands~\token_to_str:N
9954      \UnderBrace\ and~\token_to_str:N \OverBrace.\\
9955      It~will~be~ignored. \\
9956      \c_@@_available_keys_str
9957    }
9958    {
9959      The~available~keys~are~(in~alphabetic~order):~color,~left-shorten,~
9960      right-shorten,~shorten~(which~fixes~both~left-shorten~and~
9961      right-shorten)~and~yshift.
9962    }
9963  \@@_msg_new:nnn { Unknown~key~for~CodeAfter }
9964    {
9965      Unknown~key.\\
9966      The~key~'\l_keys_key_str'~is~unknown.\\
9967      It~will~be~ignored. \\
9968      \c_@@_available_keys_str
9969    }
9970    {
9971      The~available~keys~are~(in~alphabetic~order):~
9972      delimiters/color,~
9973      rules~(with~the~subkeys~'color'~and~'width'),~
9974      sub-matrix~(several~subkeys)~
9975      and~xdots~(several~subkeys).~
9976      The~latter~is~for~the~command~\token_to_str:N \line.
9977    }
9978  \@@_msg_new:nnn { Unknown~key~for~CodeBefore }
9979    {
9980      Unknown~key.\\
9981      The~key~'\l_keys_key_str'~is~unknown.\\
9982      It~will~be~ignored. \\
9983      \c_@@_available_keys_str
9984    }
9985    {
9986      The~available~keys~are~(in~alphabetic~order):~
9987      create-cell-nodes,~
9988      delimiters/color~and~
9989      sub-matrix~(several~subkeys).
9990    }
```

```
9991  \@@_msg_new:nnn { Unknown~key~for~SubMatrix }
9992    {
9993      Unknown~key.\\
9994      The~key~'\l_keys_key_str'~is~unknown.\\
9995      That~key~will~be~ignored. \\
9996      \c_@@_available_keys_str
9997    }
9998    {
9999      The~available~keys~are~(in~alphabetic~order):~
10000      'delimiters/color',~
10001      'extra-height',~
10002      'hlines',~
10003      'hvlines',~
10004      'left-xshift',~
10005      'name',~
10006      'right-xshift',~
10007      'rules'~(with~the~subkeys~'color'~and~'width'),~
10008      'slim',~
10009      'vlines'~and~'xshift'~(which~sets~both~'left-xshift'~
10010      and~'right-xshift').\\
10011    }
10012  \@@_msg_new:nnn { Unknown~key~for~notes }
10013    {
10014      Unknown~key.\\
10015      The~key~'\l_keys_key_str'~is~unknown.\\
10016      That~key~will~be~ignored. \\
10017      \c_@@_available_keys_str
10018    }
10019    {
10020      The~available~keys~are~(in~alphabetic~order):~
10021      bottomrule,~
10022      code-after,~
10023      code-before,~
10024      detect-duplicates,~
10025      enumitem-keys,~
10026      enumitem-keys-para,~
10027      para,~
10028      label-in-list,~
10029      label-in-tabular~and~
10030      style.
10031    }
10032  \@@_msg_new:nnn { Unknown~key~for~RowStyle }
10033    {
10034      Unknown~key.\\
10035      The~key~'\l_keys_key_str'~is~unknown~for~the~command~
10036      \token_to_str:N \RowStyle. \\
10037      That~key~will~be~ignored. \\
10038      \c_@@_available_keys_str
10039    }
10040    {
10041      The~available~keys~are~(in~alphabetic~order):~
10042      'bold',~
10043      'cell-space-top-limit',~
10044      'cell-space-bottom-limit',~
10045      'cell-space-limits',~
10046      'color',~
10047      'nb-rows'~and~
10048      'rowcolor'.
10049    }
10050  \@@_msg_new:nnn { Unknown~key~for~NiceMatrixOptions }
10051    {
10052      Unknown~key.\\
10053      The~key~'\l_keys_key_str'~is~unknown~for~the~command~
```

```
10054        \token_to_str:N \NiceMatrixOptions. \\
10055        That~key~will~be~ignored. \\
10056        \c_@@_available_keys_str
10057    }
10058    {
10059      The~available~keys~are~(in~alphabetic~order):~
10060      allow-duplicate-names,~
10061      caption-above,~
10062      cell-space-bottom-limit,~
10063      cell-space-limits,~
10064      cell-space-top-limit,~
10065      code-for-first-col,~
10066      code-for-first-row,~
10067      code-for-last-col,~
10068      code-for-last-row,~
10069      corners,~
10070      custom-key,~
10071      create-extra-nodes,~
10072      create-medium-nodes,~
10073      create-large-nodes,~
10074      custom-line,~
10075      delimiters~(several~subkeys),~
10076      end-of-row,~
10077      first-col,~
10078      first-row,~
10079      hlines,~
10080      hvlines,~
10081      hvlines-except-borders,~
10082      last-col,~
10083      last-row,~
10084      left-margin,~
10085      light-syntax,~
10086      light-syntax-expanded,~
10087      matrix/columns-type,~
10088      no-cell-nodes,~
10089      notes~(several~subkeys),~
10090      nullify-dots,~
10091      pgf-node-code,~
10092      renew-dots,~
10093      renew-matrix,~
10094      respect-arraystretch,~
10095      rounded-corners,~
10096      right-margin,~
10097      rules~(with~the~subkeys~'color'~and~'width'),~
10098      small,~
10099      sub-matrix~(several~subkeys),~
10100      vlines,~
10101      xdots~(several~subkeys).
10102    }
```

For '{NiceArray}', the set of keys is the same as for {NiceMatrix} excepted that there is no l and r.

```
10103  \@@_msg_new:nnn { Unknown~key~for~NiceArray }
10104    {
10105      Unknown~key.\\
10106      The~key~'\l_keys_key_str'~is~unknown~for~the~environment~
10107      \{NiceArray\}. \\
10108      That~key~will~be~ignored. \\
10109      \c_@@_available_keys_str
10110    }
10111    {
10112      The~available~keys~are~(in~alphabetic~order):~
10113      b,~
10114      baseline,~
```

```
10115       c,~
10116       cell-space-bottom-limit,~
10117       cell-space-limits,~
10118       cell-space-top-limit,~
10119       code-after,~
10120       code-for-first-col,~
10121       code-for-first-row,~
10122       code-for-last-col,~
10123       code-for-last-row,~
10124       color-inside,~
10125       columns-width,~
10126       corners,~
10127       create-extra-nodes,~
10128       create-medium-nodes,~
10129       create-large-nodes,~
10130       extra-left-margin,~
10131       extra-right-margin,~
10132       first-col,~
10133       first-row,~
10134       hlines,~
10135       hvlines,~
10136       hvlines-except-borders,~
10137       last-col,~
10138       last-row,~
10139       left-margin,~
10140       light-syntax,~
10141       light-syntax-expanded,~
10142       name,~
10143       no-cell-nodes,~
10144       nullify-dots,~
10145       pgf-node-code,~
10146       renew-dots,~
10147       respect-arraystretch,~
10148       right-margin,~
10149       rounded-corners,~
10150       rules~(with~the~subkeys~'color'~and~'width'),~
10151       small,~
10152       t,~
10153       vlines,~
10154       xdots/color,~
10155       xdots/shorten-start,~
10156       xdots/shorten-end,~
10157       xdots/shorten~and~
10158       xdots/line-style.
10159    }
```

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).

```
10160  \@@_msg_new:nnn { Unknown~key~for~NiceMatrix }
10161    {
10162      Unknown~key.\\
10163      The~key~'\l_keys_key_str'~is~unknown~for~the~
10164      \@@_full_name_env:. \\
10165      That~key~will~be~ignored. \\
10166      \c_@@_available_keys_str
10167    }
10168    {
10169      The~available~keys~are~(in~alphabetic~order):~
10170      b,~
10171      baseline,~
10172      c,~
10173      cell-space-bottom-limit,~
10174      cell-space-limits,~
10175      cell-space-top-limit,~
```

```
10176        code-after,~
10177        code-for-first-col,~
10178        code-for-first-row,~
10179        code-for-last-col,~
10180        code-for-last-row,~
10181        color-inside,~
10182        columns-type,~
10183        columns-width,~
10184        corners,~
10185        create-extra-nodes,~
10186        create-medium-nodes,~
10187        create-large-nodes,~
10188        extra-left-margin,~
10189        extra-right-margin,~
10190        first-col,~
10191        first-row,~
10192        hlines,~
10193        hvlines,~
10194        hvlines-except-borders,~
10195        l,~
10196        last-col,~
10197        last-row,~
10198        left-margin,~
10199        light-syntax,~
10200        light-syntax-expanded,~
10201        name,~
10202        no-cell-nodes,~
10203        nullify-dots,~
10204        pgf-node-code,~
10205        r,~
10206        renew-dots,~
10207        respect-arraystretch,~
10208        right-margin,~
10209        rounded-corners,~
10210        rules~(with~the~subkeys~'color'~and~'width'),~
10211        small,~
10212        t,~
10213        vlines,~
10214        xdots/color,~
10215        xdots/shorten-start,~
10216        xdots/shorten-end,~
10217        xdots/shorten~and~
10218        xdots/line-style.
10219    }
10220  \@@_msg_new:nnn { Unknown~key~for~NiceTabular }
10221    {
10222        Unknown~key.\\
10223        The~key~'\l_keys_key_str'~is~unknown~for~the~environment~
10224        \{NiceTabular\}. \\
10225        That~key~will~be~ignored. \\
10226        \c_@@_available_keys_str
10227    }
10228    {
10229        The~available~keys~are~(in~alphabetic~order):~
10230        b,~
10231        baseline,~
10232        c,~
10233        caption,~
10234        cell-space-bottom-limit,~
10235        cell-space-limits,~
10236        cell-space-top-limit,~
10237        code-after,~
10238        code-for-first-col,~
```

```
10239        code-for-first-row,~
10240        code-for-last-col,~
10241        code-for-last-row,~
10242        color-inside,~
10243        columns-width,~
10244        corners,~
10245        custom-line,~
10246        create-extra-nodes,~
10247        create-medium-nodes,~
10248        create-large-nodes,~
10249        extra-left-margin,~
10250        extra-right-margin,~
10251        first-col,~
10252        first-row,~
10253        hlines,~
10254        hvlines,~
10255        hvlines-except-borders,~
10256        label,~
10257        last-col,~
10258        last-row,~
10259        left-margin,~
10260        light-syntax,~
10261        light-syntax-expanded,~
10262        name,~
10263        no-cell-nodes,~
10264        notes~(several~subkeys),~
10265        nullify-dots,~
10266        pgf-node-code,~
10267        renew-dots,~
10268        respect-arraystretch,~
10269        right-margin,~
10270        rounded-corners,~
10271        rules~(with~the~subkeys~'color'~and~'width'),~
10272        short-caption,~
10273        t,~
10274        tabularnote,~
10275        vlines,~
10276        xdots/color,~
10277        xdots/shorten-start,~
10278        xdots/shorten-end,~
10279        xdots/shorten~and~
10280        xdots/line-style.
10281      }
10282  \@@_msg_new:nnn { Duplicate~name }
10283      {
10284        Duplicate~name.\\
10285        The~name~'\l_keys_value_tl'~is~already~used~and~you~shouldn't~use~
10286        the~same~environment~name~twice.~You~can~go~on,~but,~
10287        maybe,~you~will~have~incorrect~results~especially~
10288        if~you~use~'columns-width=auto'.~If~you~don't~want~to~see~this~
10289        message~again,~use~the~key~'allow-duplicate-names'~in~
10290        '\token_to_str:N \NiceMatrixOptions'.\\
10291        \bool_if:NF \g_@@_messages_for_Overleaf_bool
10292          { For~a~list~of~the~names~already~used,~type~H~<return>. }
10293      }
10294      {
10295        The~names~already~defined~in~this~document~are:~
10296        \seq_use:Nnnn \g_@@_names_seq { ~and~ } { ,~ } { ~and~ }.
10297      }
10298  \@@_msg_new:nn { Option~auto~for~columns-width }
10299      {
10300        Erroneous~use.\\
10301        You~can't~give~the~value~'auto'~to~the~key~'columns-width'~here.~
```

```
10302        That~key~will~be~ignored.
10303      }
10304 \@@_msg_new:nn { NiceTabularX~without~X }
10305      {
10306        NiceTabularX~without~X.\\
10307        You~should~not~use~{NiceTabularX}~without~X~columns.\\
10308        However,~you~can~go~on.
10309      }
10310 \@@_msg_new:nn { Preamble~forgotten }
10311      {
10312        Preamble~forgotten.\\
10313        You~have~probably~forgotten~the~preamble~of~your~
10314        \@@_full_name_env:. \\
10315        This~error~is~fatal.
10316      }
```

Contents