# The code of the package nicematrix[*]

F. Pantigny
fpantigny@wanadoo.fr

February 9, 2026

**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 translation: 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 registered for this package.
See: http://mirrors.ctan.org/macros/latex/contrib/l3kernel/l3prefixes.pdf
<@@=nicematrix>

First, we load pgfcore and the module shapes. We do so because it's not possible to use \usepgfmodule in \ExplSyntaxOn.

```
\RequirePackage{pgfcore}
\usepgfmodule{shapes}
```

We give the traditional declaration of a package written with the L3 programming layer.

```
\ProvidesExplPackage
  {nicematrix}
  {\myfiledate}
  {\myfileversion}
  {Enhanced arrays with the help of PGF/TikZ}
\msg_new:nnn { nicematrix } { latex-too-old }
  {
    Your~LaTeX~release~is~too~old. \\
    You~need~at~least~the~version~of~2025-06-01. \\
    If~you~use~Overleaf,~you~need~at~least~"TeXLive~2025".\\
    The~package~'nicematrix'~won't~be~loaded.
  }
\providecommand { \IfFormatAtLeastTF } { \@ifl@t@r \fmtversion }
\IfFormatAtLeastTF
  { 2025-06-01 }
  { }
  { \msg_critical:nn { nicematrix } { latex-too-old } }
```

[*]This document corresponds to the version 7.6 of nicematrix, at the date of 2026/02/01.

The command for the treatment of the options of `\usepackage` is at the end of this package for technical reasons.

```
\RequirePackage { amsmath }
```

```
\RequirePackage{array}[=2025/06/08] % v2.6j
```

```
\cs_new_protected:Npn \@@_error:n { \msg_error:nn { nicematrix } }
\cs_new_protected:Npn \@@_warning:n { \msg_warning:nn { nicematrix } }
\cs_new_protected:Npn \@@_error:nn { \msg_error:nnn { nicematrix } }
\cs_generate_variant:Nn \@@_error:nn { n e }
\cs_new_protected:Npn \@@_error:nnn { \msg_error:nnnn { nicematrix } }
\cs_new_protected:Npn \@@_fatal:n { \msg_fatal:nn { nicematrix } }
\cs_new_protected:Npn \@@_fatal:nn { \msg_fatal:nnn { nicematrix } }
\cs_new_protected:Npn \@@_msg_new:nn { \msg_new:nnn { nicematrix } }
```

With Overleaf (and also in TeXPage), 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).

```
\cs_new_protected:Npn \@@_msg_new:nnn #1 #2 #3
  {
    \bool_if:NTF \g_@@_messages_for_Overleaf_bool
      { \msg_new:nnn { nicematrix } { #1 } { #2 \\ #3 } }
      {
        \msg_new:nnnn { nicematrix } { #1 } { #2 } { #3 }
```

We keep also in memory in another message the complement of information (generally the list of the available keys) in order to write it the `log` file in all circunstancies (it will be useful for the AI of some systems such as Prism).

```
        \msg_new:nnn { nicematrix } { #1~+ } { #3 }
      }
  }
```

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

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

```
\bool_new:N \g_@@_messages_for_Overleaf_bool
\bool_gset:Nn \g_@@_messages_for_Overleaf_bool
  {
       \str_if_eq_p:on \c_sys_jobname_str { _region_ }  % for Emacs
    || \str_if_eq_p:ee \c_sys_jobname_str { output }   % for Overleaf
  }
```

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

```
\hook_gput_code:nnn { begindocument / end } { . }
  { \IfPackageLoadedT { mdwtab } { \@@_fatal:n { mdwtab~loaded } } }
```

## 2 Collecting options

The following technique allows to create user commands with the ability to put an arbitrary number of [*list of (key=val)*] after the name of the command.

*Example* :
`\@@_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`).

```
\cs_new_protected:Npn \@@_collect_options:n #1
  {
    \peek_meaning:NTF [
      { \@@_collect_options:nw { #1 } }
      { #1 { } }
  }
```

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

```
\NewDocumentCommand \@@_collect_options:nw { m r[] }
  { \@@_collect_options:nn { #1 } { #2 } }

\cs_new_protected:Npn \@@_collect_options:nn #1 #2
  {
    \peek_meaning:NTF [
      { \@@_collect_options:nnw { #1 } { #2 } }
      { #1 { #2 } }
  }

\cs_new_protected:Npn \@@_collect_options:nnw #1#2[#3]
  { \@@_collect_options:nn { #1 } { #2 , #3 } }
```

## 3 Technical definitions

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

```
\tl_const:Nn \c_@@_c_tl { c }
\tl_const:Nn \c_@@_l_tl { l }
\tl_const:Nn \c_@@_r_tl { r }
\tl_const:Nn \c_@@_all_tl { all }
\tl_const:Nn \c_@@_dot_tl { . }
\str_const:Nn \c_@@_r_str { r }
\str_const:Nn \c_@@_c_str { c }
\str_const:Nn \c_@@_l_str { l }

\tl_const:Nn \c_@@_brace_tl { nicematrix/brace }
\tl_const:Nn \c_@@_mirrored_brace_tl { nicematrix/mirrored-brace }
```

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).

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

```
\cs_generate_variant:Nn \seq_set_split:Nnn { N o }
\cs_generate_variant:Nn \str_set:Nn { N o }
\cs_generate_variant:Nn \tl_build_put_right:Nn { N o }
\prg_generate_conditional_variant:Nnn \clist_if_in:Nn { N e } { T , F, TF }
\prg_generate_conditional_variant:Nnn \tl_if_empty:n { e } { T }
\prg_generate_conditional_variant:Nnn \tl_if_head_eq_meaning:nN { o N } { TF }
\cs_generate_variant:Nn \dim_min:nn { v }
\cs_generate_variant:Nn \dim_max:nn { v }
```

```
\hook_gput_code:nnn { begindocument } { . }
  {
    \IfPackageLoadedTF { tikz }
      {
```

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`-`\endtikpicture` (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.

```
        \tl_const:Nn \c_@@_pgfortikzpicture_tl { \exp_not:N \tikzpicture }
        \tl_const:Nn \c_@@_endpgfortikzpicture_tl { \exp_not:N \endtikzpicture }
      }
      {
        \tl_const:Nn \c_@@_pgfortikzpicture_tl { \exp_not:N \pgfpicture }
        \tl_const:Nn \c_@@_endpgfortikzpicture_tl { \exp_not:N \endpgfpicture }
      }
  }
```

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 2025, the current version revtex4-2 is 4.2f (compatible with booktabs).

```
\IfClassLoadedTF { revtex4-1 }
  { \bool_const:Nn \c_@@_revtex_bool { \c_true_bool } }
  {
    \IfClassLoadedTF { revtex4-2 }
      { \bool_const:Nn \c_@@_revtex_bool { \c_true_bool } }
      {
```

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

```
        \cs_if_exist:NT \rvtx@ifformat@geq
          { \bool_const:Nn \c_@@_revtex_bool { \c_true_bool } }
          { \bool_const:Nn \c_@@_revtex_bool { \c_false_bool } }
      }
  }
```

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.

```
\cs_new_protected:Npn \@@_provide_pgfsyspdfmark:
  {
    \iow_now:Nn \@mainaux
      {
        \ExplSyntaxOn
        \cs_if_free:NT \pgfsyspdfmark
          { \cs_set_eq:NN \pgfsyspdfmark \@gobblethree }
        \ExplSyntaxOff
      }
    \cs_gset_eq:NN \@@_provide_pgfsyspdfmark: \prg_do_nothing:
  }
```

We define a command `\iddots` similar to `\ddots` ($\ddots$) but with dots going forward ($\iddots$). We use `\ProvideDocumentCommand` and so, if the command `\iddots` has already been defined (for example by the package mathdots), we don't define it again.

```
\ProvideDocumentCommand \iddots { }
  {
    \mathinner
      {
        \mkern 1 mu
        \box_move_up:nn { 1 pt } { \hbox { . } }
        \mkern 2 mu
        \box_move_up:nn { 4 pt } { \hbox { . } }
        \mkern 2 mu
        \box_move_up:nn { 7 pt }
          { \vbox:n { \kern 7 pt \hbox { . } } }
        \mkern 1 mu
      }
  }
```

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.

```
\hook_gput_code:nnn { begindocument } { . }
  {
    \IfPackageLoadedT { booktabs }
      {
        \iow_now:Nn \@mainaux
          {
            \ExplSyntaxOn
            \cs_if_exist_use:NT \nicematrix@redefine@check@rerun
            \ExplSyntaxOff
          }
      }
  }
\cs_set_protected:Npn \nicematrix@redefine@check@rerun
  {
    \let \@@_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).

```
    \cs_set_protected:Npn \pgfutil@check@rerun ##1 ##2
      {
```

`\str_if_eq:ee(TF)` is slightly faster than `\str_if_eq:nn(TF)`.

```
        \str_if_eq:eeF { nm- } { \tl_range:nnn { ##1 } { 1 } { 3 } }
          { \@@_old_pgfutil@check@rerun { ##1 } { ##2 } }
      }
  }
```

We have to know whether colortbl is loaded in particular for the redefinition of `\everycr`. The command `\@@_everycr:` will be used only in `\@@_some_initialization:`, itself in `\ar@ialign`.

```
\hook_gput_code:nnn { begindocument } { . }
  {
    \cs_set_protected:Npe \@@_everycr:
      {
        \IfPackageLoadedTF { colortbl } { \CT@everycr } { \everycr }
          { \noalign { \@@_in_everycr: } }
      }
    \IfPackageLoadedTF { colortbl }
      {
        \cs_new_eq:NN \@@_old_cellcolor: \cellcolor
        \cs_new_eq:NN \@@_old_rowcolor: \rowcolor
        \cs_new_protected:Npn \@@_revert_colortbl:
          {
            \hook_gput_code:nnn { env / tabular / begin } { nicematrix }
              {
                \cs_set_eq:NN \cellcolor \@@_old_cellcolor:
                \cs_set_eq:NN \rowcolor \@@_old_rowcolor:
              }
          }
```

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).

```
        \cs_new_protected:Npn \@@_replace_columncolor:
          {
            \tl_replace_all:Nnn \g_@@_array_preamble_tl
              { \columncolor }
              { \@@_columncolor_preamble }
```

`\@@_column_preamble`, despite its name, will be defined with `\NewDocumentCommand` because it takes in an optional argument between square brackets in first position for the colorimetric space.

```
          }
      }
      {
        \cs_new_protected:Npn \@@_revert_colortbl: { }
        \cs_new_protected:Npn \@@_replace_columncolor:
          { \cs_set_eq:NN \columncolor \@@_columncolor_preamble }
```

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.

```
        \def \CT@arc@ { }
        \def \arrayrulecolor #1 # { \CT@arc { #1 } }
        \def \CT@arc #1 #2
          {
            \dim_compare:nNnT { \baselineskip } = { \c_zero_dim } { \noalign }
              { \cs_gset_nopar:Npn \CT@arc@ { \color #1 { #2 } } }
          }
```

Idem for `\CT@drs@`.

```
        \def \doublerulesepcolor #1 # { \CT@drs { #1 } }
        \def \CT@drs #1 #2
          {
            \dim_compare:nNnT { \baselineskip } = { \c_zero_dim } { \noalign }
              { \cs_gset:Npn \CT@drsc@ { \color #1 { #2 } } }
          }
        \def \hline
          {
            \noalign { \ifnum 0 = `} \fi
            \cs_set_eq:NN \hskip \vskip
            \cs_set_eq:NN \vrule \hrule
            \cs_set_eq:NN \@width \@height
            { \CT@arc@ \vline }
            \futurelet \reserved@a
            \@xhline
          }
      }
  }
```

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.

```
\cs_set_nopar:Npn \@@_standard_cline: #1 { \@@_standard_cline:w #1 \q_stop }
\cs_set_nopar:Npn \@@_standard_cline:w #1-#2 \q_stop
  {
    \int_if_zero:nT { \l_@@_first_col_int } { \omit & }
    \int_compare:nNnT { #1 } > { \c_one_int }
      { \multispan { \int_eval:n { #1 - 1 } } & }
    \multispan { \int_eval:n { #2 - #1 + 1 } }
    {
      \CT@arc@
      \leaders \hrule \@height \arrayrulewidth \hfill
```

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

```
      \skip_horizontal:N \c_zero_dim
    }
```

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`.

```
    \everycr { }
    \cr
    \noalign { \skip_vertical:n { - \arrayrulewidth } }
  }
```

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.

```
\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`.

```
  { \@@_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*.

```
\cs_set:Npn \@@_cline_i:nn #1 #2 { \@@_cline_i:w #1|#2- \q_stop }
\cs_generate_variant:Nn \@@_cline_i:nn { e }
\cs_set:Npn \@@_cline_i:w #1|#2-#3 \q_stop
  {
    \tl_if_empty:nTF { #3 }
      { \@@_cline_iii:w #1|#2-#2 \q_stop }
      { \@@_cline_ii:w #1|#2-#3 \q_stop }
  }
\cs_set:Npn \@@_cline_ii:w #1|#2-#3- \q_stop
  { \@@_cline_iii:w #1|#2-#3 \q_stop }
\cs_set:Npn \@@_cline_iii:w #1|#2-#3 \q_stop
  {
```

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).

```
    \int_compare:nNnT { #1 } < { #2 }
      { \multispan { \int_eval:n { #2 - #1 } } & }
    \multispan { \int_eval:n { #3 - #2 + 1 } }
      {
        \CT@arc@
        \leaders \hrule \@height \arrayrulewidth \hfill
        \skip_horizontal:N \c_zero_dim
      }
```

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

```
    \peek_meaning_remove_ignore_spaces:NTF \cline
      { & \@@_cline_i:en { \int_eval:n { #3 + 1 } } }
      { \everycr { } \cr }
  }
```

[1]See question 99041 on TeX StackExchange.

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

```
\cs_set:Nn \@@_math_toggle: { $ } % $
```

```
\cs_new_protected:Npn \@@_set_CTarc:n #1
  {
    \tl_if_blank:nF { #1 }
      {
        \tl_if_head_eq_meaning:nNTF { #1 } [
          { \def \CT@arc@ { \color #1 } }
          { \def \CT@arc@ { \color { #1 } } }
      }
  }
\cs_generate_variant:Nn \@@_set_CTarc:n { o }
```

```
\cs_new_protected:Npn \@@_set_CTdrsc:n #1
  {
    \tl_if_head_eq_meaning:nNTF { #1 } [
      { \def \CT@drsc@ { \color #1 } }
      { \def \CT@drsc@ { \color { #1 } } }
  }
```

The following command must *not* be protected since it will be used to write instructions in the `\g_@@_pre_code_before_tl`.

```
\cs_new:Npn \@@_exp_color_arg:Nn #1 #2
  {
    \tl_if_head_eq_meaning:nNTF { #2 } [
      { #1 #2 }
      { #1 { #2 } }
  }
\cs_generate_variant:Nn \@@_exp_color_arg:Nn { N o }
```

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

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

```
\cs_new_protected:Npn \@@_rescan_for_spanish:N #1
  {
    \tl_set_rescan:Nno
      #1
      {
        \char_set_catcode_other:N >
        \char_set_catcode_other:N <
      }
      #1
  }
```

The L3 programming layer provides scratch dimensions `\l_tmpa_dim` and `\l_tmpb_dim`. We create several more in the same spirit.

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

```
\tl_new:N \l_@@_tmpc_tl
\tl_new:N \l_@@_tmpd_tl
```

```
\int_new:N \l_@@_tmpc_int
```

## 4 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.

```
\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).

```
\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.

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

The array will be composed in a box (named `\l_@@_the_array_box`) because we have to do manipulations concerning the potential exterior rows.

```
\box_new:N \l_@@_the_array_box
```

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

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

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

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

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

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

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

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

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

```
\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.

```
\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).

```
\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.).

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

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

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

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

```
\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.

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

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

Idem for the mono-row blocks.

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

```
\dim_new:N \l_@@_width_dim
```

The clist `\g_@@_names_clist` 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`.

```
\clist_new:N \g_@@_names_clist
```

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.

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

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

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

```
\bool_new:N \l_@@_initial_open_bool
\bool_new:N \l_@@_final_open_bool
\bool_new:N \l_@@_Vbrace_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.

```
\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).

```
\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).

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

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

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

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

```
\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 (the `X` columns of nicematrix are inspired by those of tabularx). You will use that flag for the blocks.

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

`\l_@@_V_of_X_bool` during the construction of the preamble when a column of type `X` uses the key `V` (whose name is inspired by the columns `V` of the extension varwidth).

```
\bool_new:N \l_@@_V_of_X_bool
```

The flag `g_@@_V_of_X_bool` will be raised when there is at least in the tabular a column of type `X` using the key `V`.

```
\bool_new:N \g_@@_V_of_X_bool
```

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

The following boolean will be raised when the key `no-cell-nodes` is used.

```
\bool_new:N \l_@@_no_cell_nodes_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 { g_@@_ \int_use:N \g_@@_env_int _ tl }`).

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

During the second run, if information concerning the current environment has been found in the `aux` file, the following flag will be raised. It will be used, for instance to disable several constructions (continuous dotted lines, and colored backgrounds) during the first compilation (in order to speed up it).

```
\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 information about the size of the array.

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

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

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

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

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

For `\multicolumn`.

```
\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`.

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

```
\tl_new:N \l_@@_xdots_down_tl
\tl_new:N \l_@@_xdots_up_tl
\tl_new:N \l_@@_xdots_middle_tl
```

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

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

```
\cs_new_protected:Npn \@@_test_if_math_mode:
  {
    \if_mode_math: \else:
      \@@_fatal:n { Outside~math~mode }
    \fi:
  }
```

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 analysis of the preamble of the array.

```
\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".

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

```
\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*.

```
\str_new:N \g_@@_com_or_env_str
\str_gset:Nn \g_@@_com_or_env_str { environment }
```

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

```
\clist_new:N \g_@@_cbic_clist
```

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:eeTF` and not `\tl_if_eq:eeTF` because we need to be fully expandable). `\str_if_eq:ee(TF)` is faster than `\str_if_eq:nn(TF)`.

```
\cs_new:Npn \@@_full_name_env:
  {
    \str_if_eq:eeTF { \g_@@_com_or_env_str } { command }
      { command \space \c_backslash_str \g_@@_name_env_str }
      { environment \space \{ \g_@@_name_env_str \} }
  }
```

```
\tl_new:N \g_@@_cell_after_hook_tl
```

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

```
\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.

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

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

```
\tl_new:N \g_@@_pre_code_before_tl
\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 split in two parts because we want to control the order of execution of some instructions.

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

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

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

```
\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.

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

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

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

```
\tl_new:N \l_@@_rules_color_tl
```

The sum of the weights of all the X-columns in the preamble.

```
\fp_new:N \g_@@_total_X_weight_fp
```

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 1.0 (the width of a column of weight $x$ will be that dimension multiplied by $x$). That value is computed after the construction of the array during the first compilation in order to be used in the following run.

```
\bool_new:N \l_@@_X_columns_aux_bool
\dim_new:N \l_@@_X_columns_dim
```

```
\dim_new:N \l_@@_brace_shift_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.

```
\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).

```
\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).

```
\bool_new:N \g_@@_not_empty_cell_bool
```

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

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

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

```
\dim_new:N \l_@@_x_initial_dim
\dim_new:N \l_@@_y_initial_dim
\dim_new:N \l_@@_x_final_dim
\dim_new:N \l_@@_y_final_dim
```

```
\dim_new:N \g_@@_dp_row_zero_dim
\dim_new:N \g_@@_ht_row_zero_dim
\dim_new:N \g_@@_ht_row_one_dim
\dim_new:N \g_@@_dp_ante_last_row_dim
\dim_new:N \g_@@_ht_last_row_dim
\dim_new:N \g_@@_dp_last_row_dim
```

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

```
\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".

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

```
\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!).

```
\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).

In the `\CodeBefore`, the value of `\g_@@_pos_of_blocks_seq` will be the value read in the `aux` file from a previous run. However, in the `\CodeBefore`, the commands `\EmptyColumn` and `\EmptyRow` will write virtual positions of blocks in the following sequence.

```
\seq_new:N \g_@@_future_pos_of_blocks_seq
```

They will be added to `\g_@@_pos_of_blocks_seq` after the computation of the "empty corners".

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*`}`.

```
\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.

```
\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 list. We use a `clist` instead of a `seq` because we will frequently search in that list (and searching in a `clist` is faster than searching in a `seq`).

```
\clist_new:N \l_@@_corners_cells_clist
```

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`).

```
\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.

```
\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 $n$) correspondent will be stored. These lists will be used for the creation of the "medium nodes" (if they are created).

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

By default, the diagonal lines will be parallelized[2]. 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.

```
\int_new:N \g_@@_ddots_int
\int_new:N \g_@@_iddots_int
```

The dimensions `\g_@@_delta_x_one_dim` and `\g_@@_delta_y_one_dim` will contain the $\Delta_x$ and $\Delta_y$ 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 $\Delta_x$ and $\Delta_y$ of the first `\Iddots` diagonal.

```
\dim_new:N \g_@@_delta_x_one_dim
\dim_new:N \g_@@_delta_y_one_dim
\dim_new:N \g_@@_delta_x_two_dim
\dim_new:N \g_@@_delta_y_two_dim
```

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`).

```
\int_new:N \l_@@_row_min_int
\int_new:N \l_@@_row_max_int
\int_new:N \l_@@_col_min_int
\int_new:N \l_@@_col_max_int
```

```
\int_new:N \l_@@_initial_i_int
\int_new:N \l_@@_initial_j_int
\int_new:N \l_@@_final_i_int
\int_new:N \l_@@_final_j_int
```

The following counters will be used when drawing the rules.

```
\int_new:N \l_@@_start_int
\int_set_eq:NN \l_@@_start_int \c_one_int
\int_new:N \l_@@_end_int
\int_new:N \l_@@_local_start_int
\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.

```
\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).

```
\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`.

```
\tl_new:N \l_@@_fill_tl
\tl_new:N \l_@@_opacity_tl
\tl_new:N \l_@@_draw_tl
\seq_new:N \l_@@_tikz_seq
\clist_new:N \l_@@_borders_clist
\dim_new:N \l_@@_rounded_corners_dim
```

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

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.

```
\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`.

```
\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.

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

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

```
\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`).

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

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

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

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

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

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

```
\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).

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

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

```
\dim_new:N \l_@@_submatrix_extra_height_dim
\dim_new:N \l_@@_submatrix_left_xshift_dim
\dim_new:N \l_@@_submatrix_right_xshift_dim
\clist_new:N \l_@@_hlines_clist
\clist_new:N \l_@@_vlines_clist
\clist_new:N \l_@@_submatrix_hlines_clist
\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}`).

```
\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.

```
\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`).

```
\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.

```
    \int_new:N \l_@@_first_row_int
    \int_set_eq:NN \l_@@_first_row_int \c_one_int
```

- **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.

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

```
    \int_new:N \l_@@_last_row_int
    \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”.[3]

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

  Idem for `\l_@@_last_col_without_value_bool`

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

[3]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.

- **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.

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

```
    \bool_new:N \g_@@_last_col_found_bool
```

  This boolean is set to `false` at the end of `\@@_pre_array_after_CodeBefore:`.

  In the last column, we will raise the following flag (it will be used by `\OnlyMainNiceMatrix`).

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

**Some utilities**

```
\cs_new_protected:Npn \@@_cut_on_hyphen:w #1-#2 \q_stop
  {
```

Here, we use `\def` instead of `\tl_set:Nn` for efficiency only.

```
    \def \l_tmpa_tl { #1 }
    \def \l_tmpb_tl { #2 }
  }
```

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. The second argument is `\c@iRow` or `\c@jCol`.

```
\cs_new_protected:Npn \@@_expand_clist_hvlines:NN #1 #2
  {
    \clist_if_in:NnF #1 { all }
      {
        \clist_clear:N \l_tmpa_clist
        \clist_map_inline:Nn #1
          {
            \tl_if_head_eq_meaning:nNTF { ##1 } -
              {
```

If we have yet the number of columns or the number of columns (because they have been computed during a previous run and written on the `aux` file), we can compute the actual position of the rule with a negative position.

```
                \int_if_zero:nF { #2 }
                  {
                    \clist_put_right:Ne \l_tmpa_clist
                      { \int_eval:n { #2 + (##1) + 1 } }
                  }
              }
              {
```

We recall than `\tl_if_in:nnTF` is slightly faster than `\str_if_in:nnTF`.

```
                \tl_if_in:nnTF { ##1 } { - }
                  { \@@_cut_on_hyphen:w ##1 \q_stop }
                  {
```

Here, we use `\def` instead of `\tl_set:Nn` for efficiency only.

```
                    \def \l_tmpa_tl { ##1 }
                    \def \l_tmpb_tl { ##1 }
                  }
                \int_step_inline:nnn { \l_tmpa_tl } { \l_tmpb_tl }
                  { \clist_put_right:Nn \l_tmpa_clist { ####1 } }
              }
          }
        \tl_set_eq:NN #1 \l_tmpa_clist
      }
  }
```

The following internal parameters are for:

- `\Ldots` *with both extremities open* (and hence also `\Hdotsfor` in an exterior row;
- 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.

```
\hook_gput_code:nnn { begindocument } { . }
  {
    \dim_const:Nn \c_@@_shift_Ldots_last_row_dim { 0.5 em }
    \dim_const:Nn \c_@@_innersep_middle_dim { 0.17 em }
  }
```

## 5 The command \tabularnote

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

- The caption may of course be provided by the command `\caption` in a floating environment. Of course, a command `\tabularnote` in that `\caption` makes sens only if the `\caption` is *before* the `{tabular}`.
- It's also possible to use `\tabularnote` in the value of the key `caption` of the `{NiceTabular}` when the key `caption-above` is in force. However, in that case, one must remind that the caption is composed *after* the composition of the box which contains the main tabular (that's mandatory since that caption must be wrapped with a line width equal to the width of the tabular). However, we want the labels of the successive tabular notes in the logical order. That's why:
  - The number of tabular notes present in the caption will be written on the `aux` file and available in `\g_@@_notes_caption_int`.[4]
  - 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 `\NoValue`).

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

– 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.

```
\newcounter { tabularnote }
```

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

```
\int_new:N \g_@@_tabularnote_int
\cs_set:Npn \theHtabularnote { \int_use:N \g_@@_tabularnote_int }
\seq_new:N \g_@@_notes_seq
\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.

```
\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.

```
\seq_new:N \l_@@_notes_labels_seq
\newcounter { nicematrix_draft }
\cs_new_protected:Npn \@@_notes_format:n #1
  {
    \setcounter { nicematrix_draft } { #1 }
    \@@_notes_style:n { nicematrix_draft }
  }
```

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

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

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

```
\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`.

```
\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`.

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

The tabular notes will be available for the final user only when enumitem is loaded. Indeed, the tabular notes will be composed at the end of the array with a list customized by enumitem (a list `tabularnotes` in the general case and a list `tabularnotes*` if the key `para` is in force). However, we can test whether enumitem has been loaded only at the beginning of the document (we want to allow the user to load enumitem after nicematrix).

```
\hook_gput_code:nnn { begindocument } { . }
  {
    \IfPackageLoadedTF { enumitem }
      {
```

The type of list `tabulrnotes` 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.

```
        \newlist { tabularnotes } { enumerate } { 1 }
        \setlist [ tabularnotes ]
          {
            topsep = \c_zero_dim ,
            noitemsep ,
            leftmargin = * ,
            align = left ,
            labelsep = \c_zero_dim ,
            label =
              \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotesi } } ,
          }
        \newlist { tabularnotes* } { enumerate* } { 1 }
        \setlist [ tabularnotes* ]
          {
            afterlabel = \nobreak ,
            itemjoin = \quad ,
            label =
              \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotes*i } }
          }
```

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.

```
        \NewDocumentCommand { \tabularnote } { o m }
          {
            \bool_lazy_or:nnT { \cs_if_exist_p:N \@captype } { \l_@@_in_env_bool }
              {
                \bool_lazy_and:nnTF { ! \l_@@_tabular_bool } { \l_@@_in_env_bool }
                  { \@@_error:n { tabularnote~forbidden } }
                  {
```

The second argument of `\@@_tabularnote_caption:nn` ou `\@@_tabularnote:nn` is provided by curryfication.

```
                    \bool_if:NTF \l_@@_in_caption_bool
                      {
                        \tl_if_novalue:nTF { #1 }
                          { \@@_tabularnote_caption:nn { #1 } }
                          { \@@_tabularnote_caption:nn { \exp_not:n { #1 } } }
                      }
                      {
                        \tl_if_novalue:nTF { #1 }
                          { \@@_tabularnote:nn { #1 } }
                          { \@@_tabularnote:nn { \exp_not:n { #1 } } }
                      }
                    { #2 }
                  }
              }
          }
      }
      {
        \NewDocumentCommand \tabularnote { o m }
          { \@@_err_enumitem_not_loaded: }
      }
  }
\cs_new_protected:Npn \@@_err_enumitem_not_loaded:
  {
    \@@_error_or_warning:n { enumitem~not~loaded }
    \cs_gset:Npn \@@_err_enumitem_not_loaded: { }
  }
```

```
\cs_new_protected:Npn \@@_test_first_novalue:nnn #1 #2 #3
  { \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 `\NoValue`) and `#2` is the mandatory argument of `\tabularnote`.

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

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).

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

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 `\NoValue`.
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`.

```
        \int_zero:N \l_tmpb_int
        \seq_map_indexed_inline:Nn \g_@@_notes_seq
          {
            \@@_test_first_novalue:nnn ##2 { \int_incr:N \l_tmpb_int }
            \tl_if_eq:nnT { { #1 } { #2 } } { ##2 }
              {
                \tl_if_novalue:nTF { #1 }
                  { \int_set_eq:NN \l_tmpa_int \l_tmpb_int }
                  { \int_set:Nn \l_tmpa_int { ##1 }  }
                \seq_map_break:
              }
          }
        \int_if_zero:nF { \l_tmpa_int }
          { \int_add:Nn \l_tmpa_int { \g_@@_notes_caption_int } }
      }
    \int_if_zero:nT { \l_tmpa_int }
      {
        \seq_gput_right:Nn \g_@@_notes_seq { { #1 } { #2 } }
        \tl_if_novalue:nT { #1 } { \int_gincr:N \c@tabularnote }
      }
    \seq_put_right:Ne \l_@@_notes_labels_seq
      {
        \tl_if_novalue:nTF { #1 }
          {
            \@@_notes_format:n
              {
                \int_eval:n
                  {
                    \int_if_zero:nTF { \l_tmpa_int }
                      { \c@tabularnote }
                      { \l_tmpa_int }
                  }
              }
          }
          { #1 }
     }
    \peek_meaning:NF \tabularnote
      {
```

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`.

```
        \hbox_set:Nn \l_tmpa_box
          {
```

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

```
            \@@_notes_label_in_tabular:n
              { \seq_use:Nn \l_@@_notes_labels_seq { , } }
          }
```

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

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

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

```
        \int_gincr:N \g_@@_tabularnote_int
        \refstepcounter { tabularnote }
        \int_compare:nNnT { \l_tmpa_int } = { \c@tabularnote }
          { \int_gincr:N \c@tabularnote }
        \seq_clear:N \l_@@_notes_labels_seq
        \bool_lazy_or:nnTF
          { \str_if_eq_p:ee \l_@@_hpos_cell_tl { c } }
          { \str_if_eq_p:ee \l_@@_hpos_cell_tl { r } }
          {
            \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).

```
            \skip_horizontal:n { \box_wd:N \l_tmpa_box }
          }
          { \box_use:N \l_tmpa_box }
      }
  }
```

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`.

```
\cs_new_protected:Npn \@@_tabularnote_caption:nn #1 #2
  {
    \bool_if:NTF \g_@@_caption_finished_bool
      {
        \int_compare:nNnT { \c@tabularnote } = { \g_@@_notes_caption_int }
          { \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`!

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

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

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

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.

```
            \bool_gset_true:N \g_@@_caption_finished_bool
            \int_gset_eq:NN \g_@@_notes_caption_int \c@tabularnote
            \int_gzero:N \c@tabularnote
          }
          { \seq_gput_right:Nn \g_@@_notes_in_caption_seq { { #1 } { #2 } } }
      }
```

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!

```
    \tl_if_novalue:nT { #1 } { \int_gincr:N \c@tabularnote }
    \seq_put_right:Ne \l_@@_notes_labels_seq
      {
        \tl_if_novalue:nTF { #1 }
          { \@@_notes_format:n { \int_use:N \c@tabularnote } }
          { #1 }
      }
    \peek_meaning:NF \tabularnote
      {
        \@@_notes_label_in_tabular:n { \seq_use:Nn \l_@@_notes_labels_seq { , } }
        \seq_clear:N \l_@@_notes_labels_seq
      }
  }
```

```
\cs_new_protected:Npn \@@_count_novalue_first:nn #1 #2
  { \tl_if_novalue:nT { #1 } { \int_gincr:N \g_@@_notes_caption_int } }
```

## 6 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.

```
\cs_new_protected:Npn \@@_pgf_rect_node:nnnnn #1 #2 #3 #4 #5
  {
    \begin { pgfscope }
    \pgfset
      {
        inner~sep = \c_zero_dim ,
        minimum~size = \c_zero_dim
      }
    \pgftransformshift { \pgfpoint { 0.5 * ( #2 + #4 ) } { 0.5 * ( #3 + #5 ) } }
    \pgfnode
      { rectangle }
      { center }
      {
        \vbox_to_ht:nn
          { \dim_abs:n { #5 - #3 } }
          {
            \vfill
            \hbox_to_wd:nn { \dim_abs:n { #4 - #2 } } { }
          }
      }
      { #1 }
      { }
    \end { pgfscope }
  }
```

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.

```
\cs_new_protected:Npn \@@_pgf_rect_node:nnn #1 #2 #3
  {
    \begin { pgfscope }
    \pgfset
      {
        inner~sep = \c_zero_dim ,
        minimum~size = \c_zero_dim
      }
    \pgftransformshift { \pgfpointscale { 0.5 } { \pgfpointadd { #2 } { #3 } } }
    \pgfpointdiff { #3 } { #2 }
    \pgfgetlastxy \l_tmpa_dim \l_tmpb_dim
    \pgfnode
      { rectangle }
      { center }
      {
        \vbox_to_ht:nn
          { \dim_abs:n \l_tmpb_dim }
          { \vfill \hbox_to_wd:nn { \dim_abs:n \l_tmpa_dim } { } }
      }
      { #1 }
      { }
    \end { pgfscope }
  }
```

## 7 The options

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

```
\tl_new:N \l_@@_caption_tl
\tl_new:N \l_@@_short_caption_tl
\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).

```
\bool_new:N \l_@@_caption_above_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`.

```
\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).

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

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

```
\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.

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

```
\dim_new:N \l_@@_xdots_shorten_start_dim
\dim_new:N \l_@@_xdots_shorten_end_dim
\hook_gput_code:nnn { begindocument } { . }
  {
    \dim_set:Nn \l_@@_xdots_shorten_start_dim { 0.3 em }
    \dim_set:Nn \l_@@_xdots_shorten_end_dim { 0.3 em }
  }
```

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.

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

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

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

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

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

```
\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`).

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

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

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

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

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.).

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

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

```
\cs_new_protected:Npn \@@_reset_arraystretch: { \def \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).

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

The following boolean corresponds to the key `create-cell-nodes` of the keyword `\CodeBefore`. When that key is used the "cell nodes" will be created before the `\CodeBefore` but, of course, they are *always* available in the main tabular and after!

```
\bool_new:N \g_@@_create_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.

```
\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".

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

```
\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`).

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

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

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

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

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

```
\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.

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

```
\keys_define:nn { nicematrix / xdots }
  {
    nullify .bool_set:N = \l_@@_nullify_dots_bool ,
    nullify .default:n = true ,
    brace-shift .dim_set:N = \l_@@_brace_shift_dim ,
    brace-shift .value_required:n = true ,
    brace-shift+ .code:n =
      \dim_add:Nn \l_@@_brace_shift_dim { #1 } ,
    brace-shift+ .value_required:n = true ,
    brace-shift~+ .meta:n = { brace-shift+ = #1 } ,
    Vbrace .bool_set:N = \l_@@_Vbrace_bool ,
    shorten-start .code:n =
      \hook_gput_code:nnn { begindocument } { . }
        { \dim_set:Nn \l_@@_xdots_shorten_start_dim { #1 } } ,
    shorten-start .value_required:n = true ,
    shorten-start+ .code:n =
      \hook_gput_code:nnn { begindocument } { . }
        { \dim_add:Nn \l_@@_xdots_shorten_start_dim { #1 } } ,
    shorten-start~+ .meta:n = { shorten-start += #1 } ,
    shorten-start+ .value_required:n = true ,
    shorten-end .code:n =
      \hook_gput_code:nnn { begindocument } { . }
        { \dim_set:Nn \l_@@_xdots_shorten_end_dim { #1 } } ,
    shorten-end .value_required:n = true ,
    shorten-end+ .code:n =
      \hook_gput_code:nnn { begindocument } { . }
        { \dim_add:Nn \l_@@_xdots_shorten_end_dim { #1 } } ,
    shorten-end+ .value_required:n = true ,
    shorten-end~+ .meta:n = { shorten-end += #1 } ,
    shorten .code:n =
      \hook_gput_code:nnn { begindocument } { . }
        {
          \dim_set:Nn \l_@@_xdots_shorten_start_dim { #1 }
          \dim_set:Nn \l_@@_xdots_shorten_end_dim { #1 }
        } ,
    shorten .value_required:n = true ,
    shorten+ .code:n =
      \hook_gput_code:nnn { begindocument } { . }
        {
          \dim_add:Nn \l_@@_xdots_shorten_start_dim { #1 }
          \dim_add:Nn \l_@@_xdots_shorten_end_dim { #1 }
        } ,
    shorten~+ .meta:n = { shorten+ = #1 } ,
    horizontal-labels .bool_set:N = \l_@@_xdots_h_labels_bool ,
    horizontal-labels .default:n = true ,
    horizontal-label .bool_set:N = \l_@@_xdots_h_labels_bool ,
    horizontal-label .default:n = true ,
    line-style .code:n =
      {
        \bool_lazy_or:nnTF
          { \cs_if_exist_p:N \tikzpicture }
          { \str_if_eq_p:nn { #1 } { standard } }
          { \tl_set:Nn \l_@@_xdots_line_style_tl { #1 } }
          { \@@_error:n { bad~option~for~line-style } }
      } ,
    line-style .value_required:n = true ,
    color .tl_set:N = \l_@@_xdots_color_tl ,
    color .value_required:n = true ,
    radius .code:n =
      \hook_gput_code:nnn { begindocument } { . }
        { \dim_set:Nn \l_@@_xdots_radius_dim { #1 } } ,
    radius .value_required:n = true ,
    inter .code:n =
      \hook_gput_code:nnn { begindocument } { . }
        { \dim_set:Nn \l_@@_xdots_inter_dim { #1 } } ,
    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 `^{...}`.

```
    down .code:n = \tl_put_right:Nn \l_@@_xdots_down_tl { #1 } ,
    up .code:n = \tl_put_right:Nn \l_@@_xdots_up_tl { #1 } ,
    middle .code:n = \tl_put_right:Nn \l_@@_xdots_middle_tl { #1 } ,
```

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

```
    draw-first .code:n = \prg_do_nothing: ,
    unknown .code:n =
      \@@_unknown_key:nn { nicematrix / xdots } { Unknown~key~for~xdots }
  }
```

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

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

```
\keys_define:nn { nicematrix / Global }
  {
    brace-shift .dim_set:N = \l_@@_brace_shift_dim ,
    brace-shift .value_required:n = true ,
    brace-shift+ .code:n =
      \dim_add:Nn \l_@@_brace_shift_dim { #1 } ,
    brace-shift+ .value_required:n = true ,
    brace-shift~+ .meta:n = { brace-shift+ = #1 } ,
    caption-above .code:n = \@@_error_or_warning:n { caption-above~in~env } ,
    show-cell-names .code = \@@_error_or_warning:n { show-cell-names } ,
    color-inside .code:n = \@@_fatal:n { key~color-inside } ,
    colortbl-like .code:n = \@@_fatal:n { key~color-inside } ,
    ampersand-in-blocks .bool_set:N = \l_@@_amp_in_blocks_bool ,
    ampersand-in-blocks .default:n = true ,
    &-in-blocks .meta:n = ampersand-in-blocks ,
    no-cell-nodes .code:n =
      \bool_set_true:N \l_@@_no_cell_nodes_bool
      \cs_set_protected:Npn \@@_node_cell:
```

```
      { \set@color \box_use_drop:N \l_@@_cell_box } ,
    no-cell-nodes .value_forbidden:n = true ,
    rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
    rounded-corners .default:n = 4 pt ,
    custom-line .code:n = \@@_custom_line:n { #1 } ,
    rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
    rules .value_required:n = true ,
    standard-cline .bool_set:N = \l_@@_standard_cline_bool ,
    standard-cline .default:n = true ,
    cell-space-top-limit .dim_set:N = \l_@@_cell_space_top_limit_dim ,
    cell-space-top-limit .value_required:n = true ,
    cell-space-top-limit+ .code:n =
      \dim_add:Nn \l_@@_cell_space_top_limit_dim { #1 } ,
    cell-space-top-limit+ .value_required:n = true ,
    cell-space-top-limit~+ .meta:n = { cell-space-top-limit+ = #1 } ,
    cell-space-bottom-limit .dim_set:N = \l_@@_cell_space_bottom_limit_dim ,
    cell-space-bottom-limit .value_required:n = true ,
    cell-space-bottom-limit+ .code:n =
      \dim_add:Nn \l_@@_cell_space_bottom_limit_dim { #1 } ,
    cell-space-bottom-limit+ .value_required:n = true ,
    cell-space-bottom-limit~+ .meta:n = { cell-space-bottom-limit+ = #1 } ,
    cell-space-limits .meta:n =
      {
        cell-space-top-limit = #1 ,
        cell-space-bottom-limit = #1 ,
      } ,
    cell-space-limits .value_required:n = true ,
    cell-space-limits+ .meta:n =
      {
        cell-space-top-limit += #1 ,
        cell-space-bottom-limit += #1 ,
      } ,
    cell-space-limits+ .value_required:n = true ,
    cell-space-limits~+ .meta:n = { cell-space-limits+ = #1 } ,
    xdots .code:n = \keys_set:nn { nicematrix / xdots } { #1 } ,
    light-syntax .code:n =
      \bool_set_true:N \l_@@_light_syntax_bool
      \bool_set_false:N \l_@@_light_syntax_expanded_bool ,
    light-syntax .value_forbidden:n = true ,
    light-syntax-expanded .code:n =
      \bool_set_true:N \l_@@_light_syntax_bool
      \bool_set_true:N \l_@@_light_syntax_expanded_bool ,
    light-syntax-expanded .value_forbidden:n = true ,
    end-of-row .tl_set:N = \l_@@_end_of_row_tl ,
    end-of-row .value_required:n = true ,

    first-col .code:n = \int_zero:N \l_@@_first_col_int ,
    first-row .code:n = \int_zero:N \l_@@_first_row_int ,
    last-row .int_set:N = \l_@@_last_row_int ,
    last-row .default:n = -1 ,

    code-for-first-col .tl_set:N = \l_@@_code_for_first_col_tl ,
    code-for-first-col .value_required:n = true ,
    code-for-first-col+ .code:n =
      { \tl_put_right:Nn \l_@@_code_for_first_col_tl { #1 } } ,
    code-for-first-col+ .value_required:n = true ,
    code-for-first-col~+ .meta:n = { code-for-first-col+ = #1 } ,

    code-for-last-col .tl_set:N = \l_@@_code_for_last_col_tl ,
    code-for-last-col .value_required:n = true ,
    code-for-last-col+ .code:n =
      { \tl_put_right:Nn \l_@@_code_for_last_col_tl { #1 } } ,
    code-for-last-col+ .value_required:n = true ,
    code-for-last-col~+ .meta:n = { code-for-last-col+ = #1 } ,

    code-for-first-row .tl_set:N = \l_@@_code_for_first_row_tl ,
    code-for-first-row .value_required:n = true ,
    code-for-first-row+ .code:n =
      { \tl_put_right:Nn \l_@@_code_for_first_row_tl { #1 } } ,
    code-for-first-row+ .value_required:n = true ,
    code-for-first-row~+ .meta:n = { code-for-first-row+ = #1 } ,

    code-for-last-row .tl_set:N = \l_@@_code_for_last_row_tl ,
    code-for-last-row .value_required:n = true ,
    code-for-last-row+ .code:n =
      { \tl_put_right:Nn \l_@@_code_for_first_col_tl { #1 } } ,
    code-for-last-row+ .value_required:n = true ,
    code-for-last-row~+ .meta:n = { code-for-last-row+ = #1 } ,

    hlines .clist_set:N = \l_@@_hlines_clist ,
    vlines .clist_set:N = \l_@@_vlines_clist ,
    hlines .default:n = all ,
    vlines .default:n = all ,
    vlines-in-sub-matrix .code:n =
      {
        \tl_if_single_token:nTF { #1 }
          {
            \tl_if_in:NnTF \c_@@_forbidden_letters_tl { #1 }
              { \@@_error:nn { Forbidden~letter } { #1 } }
```

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

```
              { \cs_set_eq:cN { @@ _ #1 : } \@@_make_preamble_vlism:n }
          }
          { \@@_error:n { One~letter~allowed } }
      } ,
    vlines-in-sub-matrix .value_required:n = true ,
    hvlines .code:n =
      {
        \bool_set_true:N \l_@@_hvlines_bool
        \tl_set_eq:NN \l_@@_vlines_clist \c_@@_all_tl
        \tl_set_eq:NN \l_@@_hlines_clist \c_@@_all_tl
      } ,
    hvlines .value_forbidden:n = true ,
    hvlines-except-borders .code:n =
      {
        \tl_set_eq:NN \l_@@_vlines_clist \c_@@_all_tl
        \tl_set_eq:NN \l_@@_hlines_clist \c_@@_all_tl
        \bool_set_true:N \l_@@_hvlines_bool
        \bool_set_true:N \l_@@_except_borders_bool
      } ,
    hvlines-except-borders .value_forbidden:n = true ,
    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.

```
    renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
    renew-dots .value_forbidden:n = true ,
    nullify-dots .bool_set:N = \l_@@_nullify_dots_bool ,
    create-medium-nodes .bool_set:N = \l_@@_medium_nodes_bool ,
    create-large-nodes .bool_set:N = \l_@@_large_nodes_bool ,
    create-extra-nodes .meta:n =
      { create-medium-nodes , create-large-nodes } ,
    left-margin .dim_set:N = \l_@@_left_margin_dim ,
    left-margin .default:n = \arraycolsep ,
    right-margin .dim_set:N = \l_@@_right_margin_dim ,
    right-margin .default:n = \arraycolsep ,
    margin .meta:n = { left-margin = #1 , right-margin = #1 } ,
    margin .default:n = \arraycolsep ,
    extra-left-margin .dim_set:N = \l_@@_extra_left_margin_dim ,
    extra-right-margin .dim_set:N = \l_@@_extra_right_margin_dim ,
    extra-margin .meta:n =
      { extra-left-margin = #1 , extra-right-margin = #1 } ,
    extra-margin .value_required:n = true ,
    respect-arraystretch .code:n =
      \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
    respect-arraystretch .value_forbidden:n = true ,
    pgf-node-code .tl_set:N = \l_@@_pgf_node_code_tl ,
    pgf-node-code .value_required:n = true
  }
```

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

```
\keys_define:nn { nicematrix / environments }
  {
    create-blocks-in-col .code:n = \@@_create_blocks_in_col:n { #1 } ,
    create-blocks-in-col .value_required:n = true ,
    corners .clist_set:N = \l_@@_corners_clist ,
    corners .default:n = { NW , SW , NE , SE } ,
    code-before .code:n =
      {
        \tl_if_empty:nF { #1 }
          {
            \tl_gput_left:Nn \g_@@_pre_code_before_tl { #1 }
            \bool_set_true:N \l_@@_code_before_bool
          }
      } ,
    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}`.

```
    c .code:n = \tl_set:Nn \l_@@_baseline_tl c ,
    t .code:n = \tl_set:Nn \l_@@_baseline_tl t ,
    b .code:n = \tl_set:Nn \l_@@_baseline_tl b ,
    baseline .tl_set:N = \l_@@_baseline_tl ,
    baseline .value_required:n = true ,
    columns-width .code:n =
```

We use `\str_if_eq:nnTF` which is slightly faster than `\tl_if_eq:nnTF` (and is expandable). `\str_if_eq:ee(TF)` is faster than `\str_if_eq:nn(TF)`.

```
      \str_if_eq:eeTF { #1 } { auto }
        { \bool_set_true:N \l_@@_auto_columns_width_bool }
        { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,
    columns-width .value_required:n = true ,
    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.

```
      \legacy_if:nF { measuring@ }
        {
          \str_set:Ne \l_@@_name_str { #1 }
          \clist_if_in:NoTF \g_@@_names_clist \l_@@_name_str
            { \@@_err_duplicate_names:n { #1 } }
            { \clist_gpush:No \g_@@_names_clist \l_@@_name_str }
        } ,
    name .value_required:n = true ,
    code-after .tl_gset:N = \g_nicematrix_code_after_tl ,
    code-after .value_required:n = true ,
  }
```

```
\cs_set:Npn \@@_err_duplicate_names:n #1
  { \@@_error:nn { Duplicate~name } { #1 } }
\keys_define:nn { nicematrix / notes }
  {
    para .bool_set:N = \l_@@_notes_para_bool ,
    para .default:n = true ,
    code-before .tl_set:N = \l_@@_notes_code_before_tl ,
    code-before .value_required:n = true ,
    code-before+ .code:n =
      \tl_put_right:Nn \l_@@_note_code_before_tl { #1 } ,
    code-before+ .value_required:n = true ,
    code-before~+ .code:n =
      \tl_put_right:Nn \l_@@_note_code_before_tl { #1 } ,
    code-before~+ .value_required:n = true ,
    code-after .tl_set:N = \l_@@_notes_code_after_tl ,
    code-after .value_required:n = true ,
    bottomrule .bool_set:N = \l_@@_notes_bottomrule_bool ,
    bottomrule .default:n = true ,
    style .cs_set:Np = \@@_notes_style:n #1 ,
    style .value_required:n = true ,
    label-in-tabular .cs_set:Np = \@@_notes_label_in_tabular:n #1 ,
    label-in-tabular .value_required:n = true ,
    label-in-list .cs_set:Np = \@@_notes_label_in_list:n #1 ,
    label-in-list .value_required:n = true ,
    enumitem-keys .code:n =
      {
        \hook_gput_code:nnn { begindocument } { . }
          {
            \IfPackageLoadedT { enumitem }
              { \setlist* [ tabularnotes ] { #1 } }
          }
      } ,
    enumitem-keys .value_required:n = true ,
    enumitem-keys-para .code:n =
      {
        \hook_gput_code:nnn { begindocument } { . }
          {
            \IfPackageLoadedT { enumitem }
              { \setlist* [ tabularnotes* ] { #1 } }
          }
      } ,
    enumitem-keys-para .value_required:n = true ,
    detect-duplicates .bool_set:N = \l_@@_notes_detect_duplicates_bool ,
    detect-duplicates .default:n = true ,
    unknown .code:n  =
      \@@_unknown_key:nn { nicematrix / notes } { Unknown~key~for~notes }
  }
\keys_define:nn { nicematrix / delimiters }
  {
    max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
    max-width .default:n = true ,
    color .tl_set:N = \l_@@_delimiters_color_tl ,
    color .value_required:n = true ,
  }
```

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

```
\keys_define:nn { nicematrix }
  {
    NiceMatrixOptions .inherit:n =
      { nicematrix / Global } ,
    NiceMatrixOptions / xdots .inherit:n = nicematrix / xdots ,
    NiceMatrixOptions / rules .inherit:n = nicematrix / rules ,
    NiceMatrixOptions / notes .inherit:n = nicematrix / notes ,
    NiceMatrixOptions / sub-matrix .inherit:n = nicematrix / sub-matrix ,
    SubMatrix / rules .inherit:n = nicematrix / rules ,
    CodeAfter / xdots .inherit:n = nicematrix / xdots ,
    CodeBefore / sub-matrix .inherit:n = nicematrix / sub-matrix ,
    CodeAfter / sub-matrix .inherit:n = nicematrix / sub-matrix ,
    NiceMatrix .inherit:n =
      {
        nicematrix / Global ,
        nicematrix / environments ,
      } ,
    NiceMatrix / xdots .inherit:n = nicematrix / xdots ,
    NiceMatrix / rules .inherit:n = nicematrix / rules ,
    NiceTabular .inherit:n =
      {
        nicematrix / Global ,
        nicematrix / environments
      } ,
    NiceTabular / xdots .inherit:n = nicematrix / xdots ,
    NiceTabular / rules .inherit:n = nicematrix / rules ,
    NiceTabular / notes .inherit:n = nicematrix / notes ,
    NiceArray .inherit:n =
      {
        nicematrix / Global ,
        nicematrix / environments ,
      } ,
    NiceArray / xdots .inherit:n = nicematrix / xdots ,
    NiceArray / rules .inherit:n = nicematrix / rules ,
    pNiceArray .inherit:n =
      {
        nicematrix / Global ,
        nicematrix / environments ,
      } ,
    pNiceArray / xdots .inherit:n = nicematrix / xdots ,
    pNiceArray / rules .inherit:n = nicematrix / rules ,
  }
```

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

```
\keys_define:nn { nicematrix / NiceMatrixOptions }
  {
    delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
    delimiters / color .value_required:n = true ,
    delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
    delimiters / max-width .default:n = true ,
    delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
    delimiters .value_required:n = true ,
    width .dim_set:N = \l_@@_width_dim ,
    width .value_required:n = true ,
    last-col .code:n =
      \tl_if_empty:nF { #1 }
        { \@@_error:n { last-col~non~empty~for~NiceMatrixOptions } }
        \int_zero:N \l_@@_last_col_int ,
    small .bool_set:N = \l_@@_small_bool ,
    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.

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

```
    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.

```
    columns-width .code:n =
```

We use `\str_if_eq:nnTF` which is slightly faster than `\tl_if_eq:nnTF`. `\str_if_eq:ee(TF)` is faster than `\str_if_eq:nn(TF)`.

```
      \str_if_eq:eeTF { #1 } { auto }
        { \@@_error:n { Option~auto~for~columns-width } }
        { \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.

```
    allow-duplicate-names .code:n =
      \cs_set:Nn \@@_err_duplicate_names:n { } ,
    allow-duplicate-names .value_forbidden:n = true ,
    notes .code:n = \keys_set:nn { nicematrix / notes } { #1 } ,
    notes .value_required:n = true ,
    sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
    sub-matrix .value_required:n = true ,
    matrix / columns-type .tl_set:N = \l_@@_columns_type_tl ,
    matrix / columns-type .value_required:n = true ,
    caption-above .bool_set:N = \l_@@_caption_above_bool ,
    caption-above .default:n = true ,
    unknown .code:n  =
      \@@_unknown_key:nn
        { nicematrix / Global , nicematrix / NiceMatrixOptions }
        { Unknown~key~for~NiceMatrixOptions }
  }
```

`\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.

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

```
\keys_define:nn { nicematrix / NiceMatrix }
  {
    last-col .code:n = \tl_if_empty:nTF { #1 }
                         {
                           \bool_set_true:N \l_@@_last_col_without_value_bool
                           \int_set:Nn \l_@@_last_col_int { -1 }
                         }
                         { \int_set:Nn \l_@@_last_col_int { #1 } } ,
    columns-type .tl_set:N = \l_@@_columns_type_tl ,
    columns-type .value_required:n = true ,
    l .meta:n = { columns-type = l } ,
    r .meta:n = { columns-type = r } ,
    delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
    delimiters / color .value_required:n = true ,
    delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
    delimiters / max-width .default:n = true ,
    delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
    delimiters .value_required:n = true ,
    small .bool_set:N = \l_@@_small_bool ,
    small .value_forbidden:n = true ,
    unknown .code:n =
      \@@_unknown_key:nn
        { nicematrix / Global , nicematrix / environments , nicematrix / NiceMatrix }
        { Unknown~key~for~NiceMatrix }
  }
```

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

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

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.

```
    small .bool_set:N = \l_@@_small_bool ,
    small .value_forbidden:n = true ,
    last-col .code:n = \tl_if_empty:nF { #1 }
                         { \@@_error:n { last-col~non~empty~for~NiceArray } }
                       \int_zero:N \l_@@_last_col_int ,
    r .code:n = \@@_error:n { r~or~l~with~preamble } ,
    l .code:n = \@@_error:n { r~or~l~with~preamble } ,
    unknown .code:n =
      \@@_unknown_key:nn
        { nicematrix / NiceArray , nicematrix / Global , nicematrix / environments}
        { Unknown~key~for~NiceArray }
  }
\keys_define:nn { nicematrix / pNiceArray }
  {
    first-col .code:n = \int_zero:N \l_@@_first_col_int ,
    last-col .code:n = \tl_if_empty:nF { #1 }
                         { \@@_error:n { last-col~non~empty~for~NiceArray } }
                       \int_zero:N \l_@@_last_col_int ,
    first-row .code:n = \int_zero:N \l_@@_first_row_int ,
    delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
    delimiters / color .value_required:n = true ,
    delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
    delimiters / max-width .default:n = true ,
    delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
    delimiters .value_required:n = true ,
    small .bool_set:N = \l_@@_small_bool ,
    small .value_forbidden:n = true ,
    r .code:n = \@@_error:n { r~or~l~with~preamble } ,
    l .code:n = \@@_error:n { r~or~l~with~preamble } ,
    unknown .code:n =
      \@@_unknown_key:nn
        { nicematrix / pNiceArray , nicematrix / Global , nicematrix / environments }
        { Unknown~key~for~NiceMatrix }
  }
```

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

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

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.

```
    width .code:n = \dim_set:Nn \l_@@_width_dim { #1 }
                    \bool_set_true:N \l_@@_width_used_bool ,
    width .value_required:n = true ,
    notes .code:n = \keys_set:nn { nicematrix / notes } { #1 } ,
    tabularnote .tl_gset:N = \g_@@_tabularnote_tl ,
    tabularnote .value_required:n = true ,
    caption .tl_set:N = \l_@@_caption_tl ,
    caption .value_required:n = true ,
    short-caption .tl_set:N = \l_@@_short_caption_tl ,
    short-caption .value_required:n = true ,
    label .tl_set:N = \l_@@_label_tl ,
    label .value_required:n = true ,
    last-col .code:n = \tl_if_empty:nF { #1 }
                          { \@@_error:n { last-col~non~empty~for~NiceArray } }
                         \int_zero:N \l_@@_last_col_int ,
    r .code:n = \@@_error:n { r~or~l~with~preamble } ,
    l .code:n = \@@_error:n { r~or~l~with~preamble } ,
    unknown .code:n =
      \@@_unknown_key:nn
        { nicematrix / NiceTabular , nicematrix / Global , nicematrix / environments }
        { Unknown~key~for~NiceTabular }
  }
```

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
```

```
\keys_define:nn { nicematrix / CodeAfter }
  {
    delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
    delimiters / color .value_required:n = true ,
    rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
    rules .value_required:n = true ,
    xdots .code:n = \keys_set:nn { nicematrix / xdots } { #1 } ,
    sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
    sub-matrix .value_required:n = true ,
    unknown .code:n = \@@_error:n { Unknown~key~for~CodeAfter }
  }
```

## 8 Important code used by {NiceArrayWithDelims}

The pseudo-environment `\@@_cell_begin:`–`\@@_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}`).

```
\cs_new_protected:Npn \@@_cell_begin:
  {
```

`\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.

```
    \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).

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

The following link only to have a better error message when `\Hline` is used in another place than the beginning of a line.

```
    \cs_set_eq:NN \Hline \@@_Hline_in_cell:
```

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

```
    \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.
Here is a version with the standard syntax of L3.

```
\int_compare:nNnT { \c@jCol } = { 1 }
  { \int_compare:nNnT \l_@@_first_col_int = { 1 } { \@@_begin_of_row: } }
```

We will use a version a little more efficient.

```
    \if_int_compare:w \c@jCol = \c_one_int
      \if_int_compare:w \l_@@_first_col_int = \c_one_int
        \@@_begin_of_row:
      \fi:
    \fi:
```

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:`.

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

The following command is nullified in the tabulars.

```
    \@@_tuning_not_tabular_begin:
```

```
    \@@_tuning_exterior_rows:
    \g_@@_row_style_tl
  }
\cs_new_protected:Npn \@@_tuning_exterior_rows: { }
```

Here is a version with the standard syntax of L3.

```
\cs_new_protected:Npn \@@_tuning_first_last_row:
  {
    \int_if_zero:nTF { \c@iRow }
      {
        \int_if_zero:nF { \c@jCol }
          {
            \l_@@_code_for_first_row_tl
            \xglobal \colorlet { nicematrix-first-row } { . }
          }
      }
      { \cs_gset_eq:NN \@@_tuning_exterior_rows: \@@_tuning_last_row:  }
  }
```

We will use a version a little more efficient.

```
\cs_new_protected:Npn \@@_tuning_first_last_row:
  {
    \if_int_compare:w \c@iRow = \c_zero_int
      \if_int_compare:w \c@jCol > \c_zero_int
        \l_@@_code_for_first_row_tl
        \xglobal \colorlet { nicematrix-first-row } { . }
      \fi:
    \else:
      \cs_gset_eq:NN \@@_tuning_exterior_rows: \@@_tuning_last_row:
    \fi:
  }
```

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

```
\cs_new_protected:Npn \@@_tuning_last_row:
  {
    \int_compare:nNnT { \c@iRow } = { \l_@@_last_row_int }
      {
        \l_@@_code_for_last_row_tl
        \xglobal \colorlet { nicematrix-last-row } { . }
      }
  }
```

We will use a version a little more efficient.

```
\cs_new_protected:Npn \@@_tuning_last_row:
  {
    \if_int_compare:w \c@iRow = \l_@@_last_row_int
      \l_@@_code_for_last_row_tl
      \xglobal \colorlet { nicematrix-last-row } { . }
    \fi:
  }
```

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

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

The following commands are nullified in the tabulars.

```
\cs_set_nopar:Npn \@@_tuning_not_tabular_begin:
  {
    \m@th
    $ % $
```

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

```
    \@@_tuning_key_small:
  }
\cs_set:Nn \@@_tuning_not_tabular_end: { $ } % $
```

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.

```
\cs_new_protected:Npn \@@_begin_of_row:
  {
    \int_gincr:N \c@iRow
    \dim_gset_eq:NN \g_@@_dp_ante_last_row_dim \g_@@_dp_last_row_dim
    \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
    \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
    \pgfpicture
    \pgfrememberpicturepositiononpagetrue
    \pgfcoordinate
      { \@@_env: - row - \int_use:N \c@iRow - base }
      { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
    \str_if_empty:NF \l_@@_name_str
      {
        \pgfnodealias
          { \l_@@_name_str - row - \int_use:N \c@iRow - base }
          { \@@_env: - row - \int_use:N \c@iRow - base }
      }
    \endpgfpicture
  }
```

Remark: If the key `create-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 information 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.
Here is a version with the standard syntax of L3.

```
\cs_new_protected:Npn \@@_update_for_first_and_last_row:
  {
    \int_if_zero:nTF { \c@iRow }
      {
        \dim_compare:nNnT
          { \box_dp:N \l_@@_cell_box } > { \g_@@_dp_row_zero_dim }
          { \dim_gset:Nn \g_@@_dp_row_zero_dim { \box_dp:N \l_@@_cell_box } }
        \dim_compare:nNnT
          { \box_ht:N \l_@@_cell_box } > { \g_@@_ht_row_zero_dim }
          { \dim_gset:Nn \g_@@_ht_row_zero_dim { \box_ht:N \l_@@_cell_box } }
      }
      {
        \int_compare:nNnT { \c@iRow } = { 1 }
          {
            \dim_compare:nNnT
              { \box_ht:N \l_@@_cell_box } > { \g_@@_ht_row_one_dim }
              { \dim_gset:Nn \g_@@_ht_row_one_dim { \box_ht:N \l_@@_cell_box } }
          }
      }
  }
```

We will use a version a little more efficient.

```
\cs_new_protected:Npn \@@_update_for_first_and_last_row:
  {
    \if_int_compare:w \c@iRow = \c_zero_int
      \if_dim:w \box_dp:N \l_@@_cell_box > \g_@@_dp_row_zero_dim
        \global \g_@@_dp_row_zero_dim = \box_dp:N \l_@@_cell_box
      \fi:
      \if_dim:w \box_ht:N \l_@@_cell_box > \g_@@_ht_row_zero_dim
        \global \g_@@_ht_row_zero_dim = \box_ht:N \l_@@_cell_box
      \fi:
    \else:
      \if_int_compare:w \c@iRow = \c_one_int
        \if_dim:w \box_ht:N \l_@@_cell_box > \g_@@_ht_row_one_dim
          \global \g_@@_ht_row_one_dim = \box_ht:N \l_@@_cell_box
        \fi:
      \fi:
    \fi:
  }
```

```
\cs_new_protected:Npn \@@_rotate_cell_box:
  {
    \box_rotate:Nn \l_@@_cell_box { 90 }
    \bool_if:NTF \g_@@_rotate_c_bool
      {
        \hbox_set:Nn \l_@@_cell_box
          {
            \m@th
            $ % $
            \vcenter { \box_use:N \l_@@_cell_box }
            $ % $
          }
      }
      {
        \int_compare:nNnT { \c@iRow } = { \l_@@_last_row_int }
          {
            \vbox_set_top:Nn \l_@@_cell_box
              {
                \vbox_to_zero:n { }
                \skip_vertical:n { - \box_ht:N \@arstrutbox + 0.8 ex }
                \box_use:N \l_@@_cell_box
              }
          }
       }
    \bool_gset_false:N \g_@@_rotate_bool
    \bool_gset_false:N \g_@@_rotate_c_bool
  }
```

Here is a version of the command `\@@_adjust_size_box:` with the syntax of standard L3.

```
\cs_new_protected:Npn \@@_adjust_size_box:
  {
    \dim_compare:nNnT { \g_@@_blocks_wd_dim } > { \c_zero_dim }
```

```
      {
        \dim_compare:nNnT \g_@@_blocks_wd_dim > { \box_wd:N \l_@@_cell_box }
          { \box_set_wd:Nn \l_@@_cell_box \g_@@_blocks_wd_dim }
        \dim_gzero:N \g_@@_blocks_wd_dim
      }
    \dim_compare:nNnT { \g_@@_blocks_dp_dim } > { \c_zero_dim }
      {
        \dim_compare:nNnT \g_@@_blocks_dp_dim > { \box_dp:N \l_@@_cell_box }
          { \box_set_dp:Nn \l_@@_cell_box \g_@@_blocks_dp_dim }
        \dim_gzero:N \g_@@_blocks_dp_dim
      }
    \dim_compare:nNnT { \g_@@_blocks_ht_dim } > { \c_zero_dim }
      {
        \dim_compare:nNnT \g_@@_blocks_ht_dim > { \box_ht:N \l_@@_cell_box }
          { \box_set_ht:Nn \l_@@_cell_box \g_@@_blocks_ht_dim }
        \dim_gzero:N \g_@@_blocks_ht_dim
      }
  }
```

Here is a version slightly more efficient.

```
\cs_set_protected:Npn \@@_adjust_size_box:
  {
    \if_dim:w \g_@@_blocks_wd_dim > \c_zero_dim
        \if_dim:w \g_@@_blocks_wd_dim > \box_wd:N \l_@@_cell_box
          \box_wd:N \l_@@_cell_box = \g_@@_blocks_wd_dim
        \fi:
        \global \g_@@_blocks_wd_dim  = \c_zero_dim
    \fi:
    \if_dim:w \g_@@_blocks_dp_dim > \c_zero_dim
        \if_dim:w \g_@@_blocks_dp_dim > \box_dp:N \l_@@_cell_box
          \box_dp:N \l_@@_cell_box = \g_@@_blocks_dp_dim
        \fi
        \global \g_@@_blocks_dp_dim = \c_zero_dim
    \fi:
    \if_dim:w \g_@@_blocks_ht_dim > \c_zero_dim
        \if_dim:w \g_@@_blocks_ht_dim > \box_ht:N \l_@@_cell_box
          \box_ht:N \l_@@_cell_box = \g_@@_blocks_ht_dim
        \fi:
        \global \g_@@_blocks_ht_dim = \c_zero_dim
    \fi:
  }
```

```
\cs_new_protected:Npn \@@_cell_end:
  {
```

The following command is nullified in the tabulars.

```
    \@@_tuning_not_tabular_end:
    \hbox_set_end:
    \@@_cell_end_i:
  }
```

```
\cs_new_protected:Npn \@@_cell_end_i:
  {
    \g_@@_cell_after_hook_tl
    \bool_if:NT \g_@@_rotate_bool { \@@_rotate_cell_box: }
    \@@_adjust_size_box:
    \box_set_ht:Nn \l_@@_cell_box
      { \box_ht:N \l_@@_cell_box + \l_@@_cell_space_top_limit_dim }
    \box_set_dp:Nn \l_@@_cell_box
      { \box_dp:N \l_@@_cell_box + \l_@@_cell_space_bottom_limit_dim }
    \@@_update_max_cell_width:
    \@@_update_for_first_and_last_row:
    \bool_if:NTF \g_@@_empty_cell_bool
      { \box_use_drop:N \l_@@_cell_box }
      {
        \bool_if:NTF \g_@@_not_empty_cell_bool
          { \@@_print_node_cell: }
          {
            \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > { \c_zero_dim }
              { \@@_print_node_cell: }
              { \box_use_drop:N \l_@@_cell_box }
          }
      }
    \int_compare:nNnT { \c@jCol } > { \g_@@_col_total_int }
      { \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
    \bool_gset_false:N \g_@@_empty_cell_bool
    \bool_gset_false:N \g_@@_not_empty_cell_bool
  }
```

Here is a version slightly more efficient.

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

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.

```
    \g_@@_cell_after_hook_tl
    \bool_if:NT \g_@@_rotate_bool { \@@_rotate_cell_box: }
    \@@_adjust_size_box:

    \box_set_ht:Nn \l_@@_cell_box
      { \box_ht:N \l_@@_cell_box + \l_@@_cell_space_top_limit_dim }
    \box_set_dp:Nn \l_@@_cell_box
      { \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").

```
    \@@_update_max_cell_width:
```

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

```
    \@@_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 difficult to determine whether a cell is empty. Up to now we use the following technique:

- 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.

```
    \bool_if:NTF \g_@@_empty_cell_bool
      { \box_use_drop:N \l_@@_cell_box }
      {
        \bool_if:NTF \g_@@_not_empty_cell_bool
          { \@@_print_node_cell: }
          {
            \if_dim:w \box_wd:N \l_@@_cell_box > \c_zero_dim
```

```
              \@@_print_node_cell:
            \else:
              \box_use_drop:N \l_@@_cell_box
            \fi:
          }
      }
    \if_int_compare:w \c@jCol  > \g_@@_col_total_int
      \global \g_@@_col_total_int = \c@jCol
    \fi:
    \global \let \g_@@_empty_cell_bool \c_false_bool
    \global \let \g_@@_not_empty_cell_bool \c_false_bool
  }
```

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

```
\cs_new_protected:Npn \@@_update_max_cell_width:
  {
    \dim_gset:Nn \g_@@_max_cell_width_dim
      { \dim_max:nn { \g_@@_max_cell_width_dim } { \box_wd:N \l_@@_cell_box } }
  }
```

We will use the following version, slightly more efficient:

```
\cs_new_protected:Npn \@@_update_max_cell_width:
  {
    \if_dim:w \box_wd:N \l_@@_cell_box > \g_@@_max_cell_width_dim
      \global \g_@@_max_cell_width_dim = \box_wd:N \l_@@_cell_box
    \fi:
  }
```

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

```
\cs_new_protected:Npn \@@_cell_end_for_w_s:
  {
    \@@_math_toggle:
    \hbox_set_end:
    \bool_if:NF \g_@@_rotate_bool
      {
        \hbox_set:Nn \l_@@_cell_box
          {
            \makebox [ \l_@@_col_width_dim ] [ s ]
              { \hbox_unpack_drop:N \l_@@_cell_box }
          }
      }
    \@@_cell_end_i:
  }
```

```
\pgfset
  {
    nicematrix / cell-node /.style =
     {
       inner~sep = \c_zero_dim ,
       minimum~width = \c_zero_dim
     }
  }
```

In the cells of a column of type `S` (of `siunitx`), we have to wrap the command `\@@_node_cell:` inside a command of `siunitx` to inforce the correct horizontal alignment. In the cells of the columns with other columns type, we don't have to do that job. That's why we create a socket with its default plug (`identity`) and a plug when we have to do the wrapping.

```
\socket_new:nn { nicematrix / siunitx-wrap } { 1 }
```

```
\socket_new_plug:nnn { nicematrix / siunitx-wrap } { active }
  {
    \use:c
      {
        __siunitx_table_align_
        \bool_if:NTF \l__siunitx_table_text_bool
          { \l__siunitx_table_align_text_tl }
          { \l__siunitx_table_align_number_tl }
        :n
      }
      { #1 }
  }
```

Now, a socket which deal with `create-cell-nodes` of the keyword `\CodeBefore`. When that key is used the "cell nodes" will be created before the `\CodeBefore` but, of course, they are *always* available in the main tabular and after!

```
\socket_new:nn { nicematrix / create-cell-nodes } { 1 }
\socket_new_plug:nnn { nicematrix / create-cell-nodes } { active }
  {
    \box_move_up:nn { \box_ht:N \l_@@_cell_box }
      \hbox:n
        {
          \pgfsys@markposition
            { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - NW }
        }
    #1
    \box_move_down:nn { \box_dp:N \l_@@_cell_box }
      \hbox:n
        {
          \pgfsys@markposition
            { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - SE }
        }
  }
```

```
\cs_new_protected:Npn \@@_print_node_cell:
  {
    \socket_use:nn { nicematrix / siunitx-wrap }
      { \socket_use:nn { nicematrix / create-cell-nodes } { \@@_node_cell: } }
  }
```

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

```
\cs_new_protected:Npn \@@_node_cell:
  {
    \pgfpicture
    \pgfsetbaseline \c_zero_dim
    \pgfrememberpicturepositiononpagetrue
    \pgfset { nicematrix / cell-node }
    \pgfnode
      { rectangle }
      { base }
      {
```

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

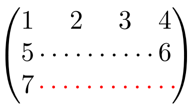
```
        \sys_if_engine_xetex:T { \set@color }
        \box_use:N \l_@@_cell_box
      }
      { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
      { \l_@@_pgf_node_code_tl }
    \str_if_empty:NF \l_@@_name_str
```

```
      {
        \pgfnodealias
          { \l_@@_name_str - \int_use:N \c@iRow - \int_use:N \c@jCol }
          { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
      }
    \endpgfpicture
  }
```

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).

```
\cs_new_protected:Npn \@@_instruction_of_type:nnn #1 #2 #3
  {
    \bool_if:nTF { #1 } { \tl_gput_left:ce } { \tl_gput_right:ce }
      { g_@@_ #2 _ lines _ tl }
      {
        \use:c { @@ _ draw _ #2 : nnn }
          { \int_use:N \c@iRow }
          { \int_use:N \c@jCol }
          { \exp_not:n { #3 } }
      }
  }
```

```
\cs_new_protected:Npn \@@_array:n
  {
    \dim_set:Nn \col@sep
      { \bool_if:NTF \l_@@_tabular_bool { \tabcolsep } { \arraycolsep } }
    \dim_compare:nNnTF { \l_@@_tabular_width_dim } = { \c_zero_dim }
      { \def \@halignto { } }
      { \cs_set_nopar:Npe \@halignto { to \dim_use:N \l_@@_tabular_width_dim } }
```

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

```
    \@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:eeTF` is fully expandable and we need something fully expandable here. `\str_if_eq:ee(TF)` is faster than `\str_if_eq:nn(TF)`.

```
    [ \str_if_eq:eeTF { \l_@@_baseline_tl } { c } { c } { t } ]
  }
\cs_generate_variant:Nn \@@_array:n { o }
```

We keep in memory the standard version of `\ar@ialign` because we will redefine `\ar@ialign` in the environment `{NiceArrayWithDelims}` but restore the standard version for use in the cells of the array. However, it seems that RevTeX goes on with a redefinition of `array` which uses `\ialign`.

```
\bool_if:NTF \c_@@_revtex_bool
  { \cs_new_eq:NN \@@_old_ialign: \ialign }
```

We use here a \cs_set_eq:cN instead of a \cs_set_eq:NN in order to avoid a message when explcheck is used on nicematrix.sty.

```
  { \cs_new_eq:cN { @@_old_ar@ialign: } \ar@ialign }
```

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

```
\cs_new_protected:Npn \@@_create_row_node:
  {
    \int_compare:nNnT { \c@iRow } > { \g_@@_last_row_node_int }
      {
        \int_gset_eq:NN \g_@@_last_row_node_int \c@iRow
        \@@_create_row_node_i:
      }
  }
\cs_new_protected:Npn \@@_create_row_node_i:
  {
```

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

```
    \hbox
      {
        \bool_if:NT \l_@@_code_before_bool
          {
            \vtop
              {
                \skip_vertical:N 0.5\arrayrulewidth
                \pgfsys@markposition
                  { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
                \skip_vertical:N -0.5\arrayrulewidth
              }
          }
        \pgfpicture
        \pgfrememberpicturepositiononpagetrue
        \pgfcoordinate { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
          { \pgfpoint \c_zero_dim { - 0.5 \arrayrulewidth } }
        \str_if_empty:NF \l_@@_name_str
          {
            \pgfnodealias
              { \l_@@_name_str - row - \int_eval:n { \c@iRow + 1 } }
              { \@@_env: - row - \int_eval:n { \c@iRow + 1 } }
          }
        \endpgfpicture
      }
  }

\cs_new_protected:Npn \@@_in_everycr:
  {
    \tbl_if_row_was_started:T { \UseTaggingSocket { tbl / row / end } }
    \tbl_update_cell_data_for_next_row:
    \int_gzero:N \c@jCol
    \bool_gset_false:N \g_@@_after_col_zero_bool
    \bool_if:NF \g_@@_row_of_col_done_bool
      {
        \@@_create_row_node:
```

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

```
        \clist_if_empty:NF \l_@@_hlines_clist
          {
            \str_if_eq:eeF { \l_@@_hlines_clist } { all }
              {
                \clist_if_in:NeT
                  \l_@@_hlines_clist
                  { \int_eval:n { \c@iRow + 1 } }
              }
              {
```

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.

```
                    \int_compare:nNnT { \c@iRow } > { -1 }
                      {
                        \int_compare:nNnF { \c@iRow } = { \l_@@_last_row_int }
                          { \hrule height \arrayrulewidth width \c_zero_dim }
                      }
                  }
              }
          }
      }
```

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

```
\cs_set_protected:Npn \@@_renew_dots:
  {
    \cs_set_eq:NN \ldots \@@_Ldots:
    \cs_set_eq:NN \cdots \@@_Cdots:
    \cs_set_eq:NN \vdots \@@_Vdots:
    \cs_set_eq:NN \ddots \@@_Ddots:
    \cs_set_eq:NN \iddots \@@_Iddots:
    \cs_set_eq:NN \dots \@@_Ldots:
    \cs_set_eq:NN \hdotsfor \@@_Hdotsfor:
  }
```

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 [5].

```
\hook_gput_code:nnn { begindocument } { . }
  {
    \IfPackageLoadedTF { booktabs }
      {
        \cs_new_protected:Npn \@@_patch_booktabs:
          { \tl_put_left:Nn \@BTnormal \@@_create_row_node_i: }
      }
      { \cs_new_protected:Npn \@@_patch_booktabs: { } }
  }
```

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`[6] 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`.

```
\cs_new_protected:Npn \@@_some_initialization:
  {
    \@@_everycr:
    \dim_gset:Nn \g_@@_dp_row_zero_dim { \box_dp:N \@arstrutbox }
    \dim_gset:Nn \g_@@_ht_row_zero_dim { \box_ht:N \@arstrutbox }
    \dim_gset_eq:NN \g_@@_ht_row_one_dim \g_@@_ht_row_zero_dim
    \dim_gzero:N \g_@@_dp_ante_last_row_dim
    \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
    \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
  }
```

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

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

`\@@_pre_array_after_CodeBefore:` will be executed in `\@@_pre_array:` *after* the execution of the `\CodeBefore`. It contains all the code before the beginning of the construction of `\l_@@_the_array_box`.

```
\cs_new_protected:Npn \@@_pre_array_after_CodeBefore:
  {
```

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 reinitialize that variable with the content of `\g_@@_future_pos_of_blocks_seq` because the mains blocks will be added in `\g_@@_pos_of_blocks_seq` during the construction of the array.

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

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

```
    \seq_gclear_new:N \g_@@_multicolumn_cells_seq
    \seq_gclear_new:N \g_@@_multicolumn_sizes_seq
```

The command `\create_row_node:` will create a row-node (and not a row of nodes!). However, at the end of the array we construct a “false row” (for the col-nodes) and it interferes with the construction of the last row-node of the array. We don’t want to create such row-node twice (to avaid warnings or, maybe, errors). That’s why the command `\@@_create_row_node:` will use the following counter to avoid such construction.

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

The value $-2$ is important.

The total weight of the letters `X` in the preamble of the array.

```
    \fp_gzero:N \g_@@_total_X_weight_fp
    \bool_gset_false:N \g_@@_V_of_X_bool
    \@@_expand_clist_hvlines:NN \l_@@_hlines_clist \c@iRow
    \@@_expand_clist_hvlines:NN \l_@@_vlines_clist \c@jCol
    \@@_patch_booktabs:
    \box_clear_new:N \l_@@_cell_box
    \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}`).

```
    \bool_if:NT \l_@@_small_bool
      {
        \def \arraystretch { 0.47 }
        \dim_set:Nn \arraycolsep { 1.45 pt }
```

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

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

The boolean `\g_@@_create_cell_nodes_bool` corresponds to the key `create-cell-nodes` of the keyword `\CodeBefore`. When that key is used the “cell nodes” will be created before the `\CodeBefore` but, of course, they are *always* available in the main tabular and after!

```
    \bool_if:NT \g_@@_create_cell_nodes_bool
      {
        \tl_put_right:Nn \@@_begin_of_row:
          {
            \pgfsys@markposition
              { \@@_env: - row - \int_use:N \c@iRow - base }
          }
        \socket_assign_plug:nn { nicematrix / create-cell-nodes } { active }
      }
```

The environment {array} uses internally the command \ar@ialign. We change that command for several reasons. In particular, \ar@ialign sets \everycr to { } and we *need* to change the value of \everycr.

```
    \bool_if:NF \c_@@_revtex_bool
      {
        \def \ar@ialign
          {
            \tbl_init_cell_data_for_table:
            \@@_some_initialization:
            \dim_zero:N \tabskip
```

After its first use, the definition of \ar@ialign will revert automatically to its default definition. With that programmation, we will have, in the cells of the array, a clean version of \ar@ialign. We use \cs_set_eq:Nc instead of \cs_set_eq:NN in order to avoid a message when explcheck is used on nicematrix.sty.

```
            \cs_set_eq:Nc \ar@ialign { @@_old_ar@ialign: }
            \halign
          }
      }
```

It seems that there is a problem when nicematrix is used with in revtex4-2 with the package colortbl loaded. The following code prevent that problem but it does *not* treat the actual problem! It's only a patch *ad hoc*.
That patch has been added in version 7.0x, 2024-11-27 (question by mail of Tamra Nebabu).

```
    \bool_if:NT \c_@@_revtex_bool
      {
        \IfPackageLoadedT { colortbl }
          { \cs_set_protected:Npn \CT@setup { } }
      }
```

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).

```
    \cs_set_eq:NN \@@_old_ldots: \ldots
    \cs_set_eq:NN \@@_old_cdots: \cdots
    \cs_set_eq:NN \@@_old_vdots: \vdots
    \cs_set_eq:NN \@@_old_ddots: \ddots
    \cs_set_eq:NN \@@_old_iddots: \iddots
    \bool_if:NTF \l_@@_standard_cline_bool
      { \cs_set_eq:NN \cline \@@_standard_cline: }
      { \cs_set_eq:NN \cline \@@_cline: }
    \cs_set_eq:NN \Ldots \@@_Ldots:
    \cs_set_eq:NN \Cdots \@@_Cdots:
    \cs_set_eq:NN \Vdots \@@_Vdots:
    \cs_set_eq:NN \Ddots \@@_Ddots:
    \cs_set_eq:NN \Iddots \@@_Iddots:
    \cs_set_eq:NN \Hline \@@_Hline:
    \cs_set_eq:NN \Hspace \@@_Hspace:
    \cs_set_eq:NN \Hdotsfor \@@_Hdotsfor:
    \cs_set_eq:NN \Vdotsfor \@@_Vdotsfor:
    \cs_set_eq:NN \Block \@@_Block:
    \cs_set_eq:NN \rotate \@@_rotate:
    \cs_set_eq:NN \OnlyMainNiceMatrix \@@_OnlyMainNiceMatrix:n
    \cs_set_eq:NN \dotfill \@@_dotfill:
    \cs_set_eq:NN \CodeAfter \@@_CodeAfter:
    \cs_if_free:NT \Body { \cs_set_eq:NN \Body \@@_Body: }
    \cs_set_eq:NN \diagbox \@@_diagbox:nn
    \cs_set_eq:NN \NotEmpty \@@_NotEmpty:
    \cs_set_eq:NN \TopRule \@@_TopRule
    \cs_set_eq:NN \MidRule \@@_MidRule
    \cs_set_eq:NN \BottomRule \@@_BottomRule
    \cs_set_eq:NN \RowStyle \@@_RowStyle:n
    \cs_set_eq:NN \Hbrace \@@_Hbrace
    \cs_set_eq:NN \Vbrace \@@_Vbrace
    \seq_map_inline:Nn \l_@@_custom_line_commands_seq
      { \cs_set_eq:cc { ##1 } { nicematrix - ##1 } }
    \cs_set_eq:NN \cellcolor \@@_cellcolor_tabular
    \cs_set_eq:NN \rowcolor \@@_rowcolor_tabular
    \cs_set_eq:NN \rowcolors \@@_rowcolors_tabular
    \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors_tabular
    \int_if_zero:nTF \l_@@_first_row_int
      { \cs_gset_eq:NN \@@_tuning_exterior_rows: \@@_tuning_first_last_row: }
      {
        \int_compare:nNnT { \l_@@_last_row_int } > { \c_zero_int }
          { \cs_gset_eq:NN \@@_tuning_exterior_rows: \@@_tuning_last_row: }
      }
    \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:`.

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

```
    \tl_if_exist:NT \l_@@_note_in_caption_tl
      {
        \tl_if_empty:NF \l_@@_note_in_caption_tl
          {
            \int_gset:Nn \g_@@_notes_caption_int { \l_@@_note_in_caption_tl }
            \int_gset:Nn \c@tabularnote { \l_@@_note_in_caption_tl }
          }
      }
```

The sequence `\g_@@_multicolumn_cells_seq` will contain the list of the cells of the array where a command `\multicolumn{`$n$`}{...}{...}` with $n > 1$ is issued. In `\g_@@_multicolumn_sizes_seq`, the "sizes" (that is to say the values of $n$) correspondent will be stored. These lists will be used for the creation of the "medium nodes" (if they are created).

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

```
    \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 of rows excepted the last row (if `\l_@@_last_row_bool` has been raised with the option `last-row`).

```
    \int_gzero: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:` executed at the beginning of each cell.

```
    \int_gzero:N \g_@@_col_total_int

    \cs_set_eq:NN \@ifnextchar \new@ifnextchar

    \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.

```
    \tl_gclear_new:N \g_@@_Cdots_lines_tl
    \tl_gclear_new:N \g_@@_Ldots_lines_tl
    \tl_gclear_new:N \g_@@_Vdots_lines_tl
    \tl_gclear_new:N \g_@@_Ddots_lines_tl
    \tl_gclear_new:N \g_@@_Iddots_lines_tl
    \tl_gclear_new:N \g_@@_HVdotsfor_lines_tl

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

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]).

```
    \dim_zero_new:N \l_@@_left_delim_dim
    \dim_zero_new:N \l_@@_right_delim_dim
    \bool_if:NTF \g_@@_delims_bool
      {
```

The command \bBigg@ is a command of amsmath.

```
        \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_left_delim_tl $ }
        \dim_set:Nn \l_@@_left_delim_dim { \box_wd:N \l_tmpa_box }
        \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_right_delim_tl $ }
        \dim_set:Nn \l_@@_right_delim_dim { \box_wd:N \l_tmpa_box }
      }
      {
        \dim_gset:Nn \l_@@_left_delim_dim
          { 2 \bool_if:NTF \l_@@_tabular_bool { \tabcolsep } { \arraycolsep } }
        \dim_gset_eq:NN \l_@@_right_delim_dim \l_@@_left_delim_dim
      }
  }
```

This is the end of \@@_pre_array_after_CodeBefore:.

The command \@@_pre_array: will be executed after analysis of the keys of the environment. If will, in particular, read the potential informations written on the aux file.

```
\cs_new_protected:Npn \@@_pre_array:
  {
    \cs_if_exist:NT \theiRow { \int_set_eq:NN \l_@@_old_iRow_int \c@iRow }
    \int_gzero_new:N \c@iRow
    \cs_if_exist:NT \thejCol { \int_set_eq:NN \l_@@_old_jCol_int \c@jCol }
    \int_gzero_new:N \c@jCol
```

We give values to the LaTeX counters iRow and jCol. We remind that before and after the main array (in particular in the \CodeBefore and 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).

```
    \int_compare:nNnT \l_@@_last_row_int > 0
      { \int_set:Nn \c@iRow { \l_@@_last_row_int - 1 } }
    \int_compare:nNnT \l_@@_last_col_int > 0
      { \int_set:Nn \c@jCol { \l_@@_last_col_int - 1 } }
    \bool_if:NT \g_@@_aux_found_bool
      {
        \int_set:Nn \c@iRow { \seq_item:Nn \g_@@_size_seq { 2 } }
        \int_set:Nn \c@jCol { \seq_item:Nn \g_@@_size_seq { 5 } }
        \int_gset:Nn \g_@@_row_total_int { \seq_item:Nn \g_@@_size_seq { 3 } }
        \int_gset:Nn \g_@@_col_total_int { \seq_item:Nn \g_@@_size_seq { 6 } }
      }
```

We recall that \l_@@_last_row_int and \l_@@_last_col_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-col (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).

```
    \int_compare:nNnT { \l_@@_last_row_int } = { -1 }
      {
        \bool_set_true:N \l_@@_last_row_without_value_bool
        \bool_if:NT \g_@@_aux_found_bool
          { \int_set:Nn \l_@@_last_row_int { \seq_item:Nn \g_@@_size_seq { 3 } } }
      }
    \int_compare:nNnT { \l_@@_last_col_int } = { -1 }
      {
        \bool_if:NT \g_@@_aux_found_bool
          { \int_set:Nn \l_@@_last_col_int { \seq_item:Nn \g_@@_size_seq { 6 } } }
      }
```

If there is an exterior row, we patch a command used in `\@@_cell_begin:` in order to keep track of some dimensions needed to the construction of that "last row".

```
    \int_compare:nNnT { \l_@@_last_row_int } > { -2 }
      {
        \tl_put_right:Nn \@@_update_for_first_and_last_row:
          {
            \dim_compare:nNnT { \g_@@_ht_last_row_dim } < { \box_ht:N \l_@@_cell_box }
              { \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \l_@@_cell_box } }
            \dim_compare:nNnT { \g_@@_dp_last_row_dim } < { \box_dp:N \l_@@_cell_box }
              { \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \l_@@_cell_box } }
          }
      }
```

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

Now the `\CodeBefore`.

```
    \bool_if:NT \l_@@_code_before_bool { \@@_exec_code_before: }
```

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

```
    \@@_pre_array_after_CodeBefore:
```

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 `$` also).

```
    \hbox_set:Nw \l_@@_the_array_box
    \skip_horizontal:N \l_@@_left_margin_dim
    \skip_horizontal:N \l_@@_extra_left_margin_dim
    \UseTaggingSocket { tbl / hmode / begin }
```

The following code is a workaround to specify to the tagging system that the following code is *fake math* (it raises `\l__math_fakemath_bool` in recent versions of LaTeX).

```
    \m@th
    $ % $
    \bool_if:NTF \l_@@_light_syntax_bool
      { \use:c { @@-light-syntax } }
      { \use:c { @@-normal-syntax } }
  }
```

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

```
\cs_new_protected_nopar:Npn \@@_CodeBefore_Body:w #1 \Body
  {
    \tl_set:Nn \l_tmpa_tl { #1 }
    \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
      { \@@_rescan_for_spanish:N \l_tmpa_tl }
    \tl_gput_left:No \g_@@_pre_code_before_tl \l_tmpa_tl
    \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...

```
    \@@_pre_array:
  }
```

## 9 The \CodeBefore

```
\cs_new_protected_nopar:Npn \@@_Body: { \@@_fatal:n { Body~alone } }
```

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

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

We will create all the `col` nodes and `row` nodes with the information written in the `aux` file. You use the technique described in the page 1247 of `pgfmanual.pdf`, version 3.1.10.

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

First, the recreation of the `row` nodes.

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

Now, the recreation of the `col` nodes.

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

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

```
    \bool_if:NT \g_@@_create_cell_nodes_bool { \@@_recreate_cell_nodes: }
    \endpgfpicture
```

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

```
    \@@_create_diag_nodes:
```

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

```
    \@@_create_blocks_nodes:
    \IfPackageLoadedT { tikz }
      {
        \tikzset
          {
            every~picture / .style =
              { overlay , name~prefix = \@@_env: - }
          }
      }
    \cs_set_eq:NN \cellcolor \@@_cellcolor
    \cs_set_eq:NN \rectanglecolor \@@_rectanglecolor
    \cs_set_eq:NN \roundedrectanglecolor \@@_roundedrectanglecolor
    \cs_set_eq:NN \rowcolor \@@_rowcolor
    \cs_set_eq:NN \rowcolors \@@_rowcolors
    \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors
    \cs_set_eq:NN \arraycolor \@@_arraycolor
    \cs_set_eq:NN \columncolor \@@_columncolor
    \cs_set_eq:NN \chessboardcolors \@@_chessboardcolors
    \cs_set_eq:NN \SubMatrix \@@_SubMatrix_in_code_before
    \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
    \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
    \cs_set_eq:NN \EmptyColumn \@@_EmptyColumn:n
    \cs_set_eq:NN \EmptyRow \@@_EmptyRow:n
  }
```

```
\cs_new_protected:Npn \@@_exec_code_before:
  {
```

We mark the cells which are in the (empty) corners because those cells must not be colored. We should try to find a way to detected whether we actually have coloring instructions to execute...

```
    \clist_map_inline:Nn \l_@@_corners_cells_clist
      { \cs_set_nopar:cpn { @@ _ corner _ ##1 } { } }
    \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.

```
    \@@_add_to_colors_seq:nn { { nocolor } } { }
    \bool_gset_false:N \g_@@_create_cell_nodes_bool
    \group_begin:
```

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

```
    \if_mode_math:
      \@@_exec_code_before_i:
    \else:
      $ % $
      \@@_exec_code_before_i:
      $ % $
    \fi:
    \group_end:
  }
```

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).

```
\cs_new_protected:Npn \@@_exec_code_before_i:
  {
    \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 }
      { \@@_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`.

```
      \exp_last_unbraced:No \@@_CodeBefore_keys:
        \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.

```
      \@@_actually_color:
      \l_@@_code_before_tl
      \q_stop
  }
```

```
\keys_define:nn { nicematrix / CodeBefore }
  {
    create-cell-nodes .bool_gset:N = \g_@@_create_cell_nodes_bool ,
    create-cell-nodes .default:n = true ,
    sub-matrix .code:n = \keys_set:nn { nicematrix / sub-matrix } { #1 } ,
    sub-matrix .value_required:n = true ,
    delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
    delimiters / color .value_required:n = true ,
    unknown .code:n = \@@_error:n { Unknown~key~for~CodeBefore }
  }
\NewDocumentCommand \@@_CodeBefore_keys: { O { } }
  {
    \keys_set:nn { nicematrix / CodeBefore } { #1 }
    \@@_CodeBefore:w
  }
```

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.

```
\cs_new_protected:Npn \@@_CodeBefore:w #1 \q_stop
  {
    \bool_if:NTF \g_@@_aux_found_bool
      {
        \@@_pre_code_before:
        \legacy_if:nF { measuring@ } { #1 }
      }
```

If we are in the first compilation, you won't really execute the `\CodeBefore` but we have to execute some instructions of creation of PGF/TikZ pictures in order to have the correct `aux` file in the next run (hence, we avoid to "lose" a run).

```
      {
        \pgfsys@markposition { \@@_env: - position }
        \pgfsys@getposition { \@@_env: - position } \@@_picture_position:
        \pgfpicture
          \pgf@relevantforpicturesizefalse
        \endpgfpicture
```

The following picture corresponds to `\@@_create_diag_nodes:`

```
        \pgfpicture
          \pgfrememberpicturepositiononpagetrue
        \endpgfpicture
```

The following picture corresponds to `\@@_create_blocks_nodes:`.

```
        \pgfpicture
          \pgf@relevantforpicturesizefalse
          \pgfrememberpicturepositiononpagetrue
        \endpgfpicture
```

The following picture corresponds `\@@_actually_color:`

```
        \pgfpicture
          \pgf@relevantforpicturesizefalse
        \endpgfpicture
      }
  }
```

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.

```
\cs_new_protected:Npn \@@_recreate_cell_nodes:
  {
    \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int }
      {
        \pgfsys@getposition { \@@_env: - ##1 - base } \@@_node_position:
        \pgfcoordinate { \@@_env: - row - ##1 - base }
          { \pgfpointdiff \@@_picture_position: \@@_node_position: }
        \int_step_inline:nnn { \l_@@_first_col_int } { \g_@@_col_total_int }
```

```
          {
            \cs_if_exist:cT
              { pgf @ sys @ pdf @ mark @ pos @ \@@_env: - ##1 - ####1 - NW }
              {
                \pgfsys@getposition
                  { \@@_env: - ##1 - ####1 - NW }
                  \@@_node_position:
                \pgfsys@getposition
                  { \@@_env: - ##1 - ####1 - SE }
                  \@@_node_position_i:
                \@@_pgf_rect_node:nnn
                  { \@@_env: - ##1 - ####1 }
                  { \pgfpointdiff \@@_picture_position: \@@_node_position: }
                  { \pgfpointdiff \@@_picture_position: \@@_node_position_i: }
              }
          }
      }
    \@@_create_extra_nodes:
    \@@_create_aliases_last:
  }
```

```
\cs_new_protected:Npn \@@_create_aliases_last:
  {
    \int_step_inline:nn { \c@iRow }
      {
        \pgfnodealias
          { \@@_env: - ##1 - last }
          { \@@_env: - ##1 - \int_use:N \c@jCol }
      }
    \int_step_inline:nn { \c@jCol }
      {
        \pgfnodealias
          { \@@_env: - last - ##1 }
          { \@@_env: - \int_use:N \c@iRow - ##1 }
      }
    \pgfnodealias
      { \@@_env: - last - last }
      { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
  }
```

```
\cs_new_protected:Npn \@@_create_blocks_nodes:
  {
    \pgfpicture
    \pgf@relevantforpicturesizefalse
    \pgfrememberpicturepositiononpagetrue
    \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
      { \@@_create_one_block_node:nnnnn ##1 }
    \endpgfpicture
  }
```

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.[7]

```
\cs_new_protected:Npn \@@_create_one_block_node:nnnnn #1 #2 #3 #4 #5
  {
    \tl_if_empty:nF { #5 }
      {
        \@@_qpoint:n { col - #2 }
        \dim_set_eq:NN \l_tmpa_dim \pgf@x
        \@@_qpoint:n { #1 }
```

[7]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).

```
        \dim_set_eq:NN \l_tmpb_dim \pgf@y
        \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
        \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
        \@@_qpoint:n { \int_eval:n { #3 + 1 } }
        \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
        \@@_pgf_rect_node:nnnnn
          { \@@_env: - #5 }
          { \dim_use:N \l_tmpa_dim }
          { \dim_use:N \l_tmpb_dim }
          { \dim_use:N \l_@@_tmpc_dim }
          { \dim_use:N \l_@@_tmpd_dim }
      }
  }
```

```
\cs_new_protected:Npn \@@_patch_for_revtex:
  {
    \cs_set_eq:NN \@addamp \@addamp@LaTeX
    \cs_set_eq:NN \@array \@array@array
    \cs_set_eq:NN \@tabular \@tabular@array
    \cs_set:Npn \@tabarray { \@ifnextchar [ { \@array } { \@array [ c ] } }
    \cs_set_eq:NN \array \array@array
    \cs_set_eq:NN \endarray \endarray@array
    \cs_set:Npn \endtabular { \endarray $\egroup} % $
    \cs_set_eq:NN \@mkpream \@mkpream@array
    \cs_set_eq:NN \@classx \@classx@array
    \cs_set_eq:NN \insert@column \insert@column@array
    \cs_set_eq:NN \@arraycr \@arraycr@array
    \cs_set_eq:NN \@xarraycr \@xarraycr@array
    \cs_set_eq:NN \@xargarraycr \@xargarraycr@array
  }
```

## 10 The environment {NiceArrayWithDelims}

```
\NewDocumentEnvironment { NiceArrayWithDelims }
  { m m O { } m ! O { } t \CodeBefore }
  {
    \bool_if:NT \c_@@_revtex_bool { \@@_patch_for_revtex: }
    \@@_provide_pgfsyspdfmark:
    \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.

```
    \bgroup

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

    \int_gzero:N \g_@@_block_box_int
    \dim_gzero:N \g_@@_width_last_col_dim
    \dim_gzero:N \g_@@_width_first_col_dim
    \bool_gset_false:N \g_@@_row_of_col_done_bool
    \str_if_empty:NT \g_@@_name_env_str
      { \str_gset:Nn \g_@@_name_env_str { NiceArrayWithDelims } }
    \bool_if:NTF \l_@@_tabular_bool
      { \mode_leave_vertical: }
      { \@@_test_if_math_mode: }
    \bool_if:NT \l_@@_in_env_bool { \@@_fatal:n { Yet~in~env } }
    \bool_set_true:N \l_@@_in_env_bool
```

The command `\CT@arc@` contains the instruction of color for the rules of the array[8]. 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.

```
    \cs_gset_eq:cN { @@_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.

```
    \cs_if_exist:NT \tikz@library@external@loaded
      {
        \tikzexternaldisable
        \cs_if_exist:NT \ifstandalone
          { \tikzset { external / optimize = false } }
      }
```

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

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

The sequence `\g_@@_blocks_seq` will contain the characteristics of the blocks (specified by `\Block`) of the array. The sequence `\g_@@_pos_of_blocks_seq` will contain only the position of the blocks.

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

```
    \seq_gclear:N \g_@@_pos_of_stroken_blocks_seq
    \seq_gclear:N \g_@@_pos_of_xdots_seq
    \tl_gclear_new:N \g_@@_code_before_tl
    \tl_gclear:N \g_@@_row_style_tl
```

We load all the information written in the `aux` file during previous compilations corresponding to the current environment.

```
    \tl_if_exist:cTF { g_@@ _ \int_use:N \g_@@_env_int _ tl }
      {
        \bool_gset_true:N \g_@@_aux_found_bool
        \use:c { g_@@ _ \int_use:N \g_@@_env_int _ tl }
      }
      { \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.

```
    \tl_gclear:N \g_@@_aux_tl
    \tl_if_empty:NF \g_@@_code_before_tl
      {
        \bool_set_true:N \l_@@_code_before_bool
        \tl_put_right:No \l_@@_code_before_tl \g_@@_code_before_tl
      }
    \tl_if_empty:NF \g_@@_pre_code_before_tl
      { \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`.

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

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

```
    \@@_set_CTarc:o \l_@@_rules_color_tl % noqa: w302
```

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:.

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

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

```
  {
    \bool_if:NTF \l_@@_light_syntax_bool
      { \use:c { end @@-light-syntax } }
      { \use:c { end @@-normal-syntax } }
    $ % $
    \skip_horizontal:N \l_@@_right_margin_dim
    \skip_horizontal:N \l_@@_extra_right_margin_dim
    \hbox_set_end:
    \UseTaggingSocket { tbl / hmode / 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.

```
    \bool_if:NT \l_@@_width_used_bool
      {
        \fp_compare:nNnT { \g_@@_total_X_weight_fp } = { \c_zero_fp }
          { \@@_error_or_warning:n { width~without~X~columns } }
      }
```

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 1.0. For a X-column of weight $x$, the width will be \l_@@_X_columns_dim multiplied by $x$.

```
    \fp_compare:nNnT { \g_@@_total_X_weight_fp } > { \c_zero_fp }
      { \@@_compute_width_X: }
```

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).

```
    \int_compare:nNnT { \l_@@_last_row_int } > { -2 }
      {
        \bool_if:NF \l_@@_last_row_without_value_bool
          {
            \int_compare:nNnF { \l_@@_last_row_int } = { \c@iRow }
              {
                \@@_error:n { Wrong~last~row }
                \int_set_eq:NN \l_@@_last_row_int \c@iRow
              }
          }
      }
```

Now, the definition of \c@jCol and \g_@@_col_total_int changes: \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".[9]

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

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

```
      }
```

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

```
    \int_gset_eq:NN \g_@@_row_total_int \c@iRow
    \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. 95).

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

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

```
    \bool_if:nTF { ! \g_@@_delims_bool }
      {
        \str_if_eq:eeTF { \l_@@_baseline_tl } { c }
          { \@@_use_arraybox_with_notes_c: }
          {
            \str_if_eq:eeTF { \l_@@_baseline_tl } { b }
              { \@@_use_arraybox_with_notes_b: }
              { \@@_use_arraybox_with_notes: }
          }
      }
```

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).

```
      {
        \int_if_zero:nTF { \l_@@_first_row_int }
          {
            \dim_set_eq:NN \l_tmpa_dim \g_@@_dp_row_zero_dim
            \dim_add:Nn \l_tmpa_dim \g_@@_ht_row_zero_dim
          }
          { \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".[10]

```
        \int_compare:nNnTF { \l_@@_last_row_int } > { -2 }
          {
            \dim_set_eq:NN \l_tmpb_dim \g_@@_ht_last_row_dim
            \dim_add:Nn \l_tmpb_dim \g_@@_dp_last_row_dim
          }
          { \dim_zero:N \l_tmpb_dim }
        \hbox_set:Nn \l_tmpa_box
          {
            \m@th
            $ % $
            \@@_color:o \l_@@_delimiters_color_tl
            \exp_after:wN \left \g_@@_left_delim_tl
            \vcenter
              {
```

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.

```
                \skip_vertical:n { - \l_tmpa_dim - \arrayrulewidth }
                \hbox
                  {
                    \bool_if:NTF \l_@@_tabular_bool
                      { \skip_horizontal:n { - \tabcolsep } }
                      { \skip_horizontal:n { - \arraycolsep } }
                    \@@_use_arraybox_with_notes_c:
                    \bool_if:NTF \l_@@_tabular_bool
```

[10]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).

```
                            { \skip_horizontal:n { - \tabcolsep } }
                            { \skip_horizontal:n { - \arraycolsep } }
                        }
```

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

```
                    \skip_vertical:n { - \l_tmpb_dim  + \arrayrulewidth }
                  }
                \exp_after:wN \right \g_@@_right_delim_tl
                $ % $
              }
```

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.

```
          \bool_if:NTF \l_@@_delimiters_max_width_bool
            {
              \@@_put_box_in_flow_bis:nn
                { \g_@@_left_delim_tl }
                { \g_@@_right_delim_tl }
            }
            \@@_put_box_in_flow:
        }
```

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. 96).

```
      \bool_if:NT \g_@@_last_col_found_bool
        { \skip_horizontal:N \g_@@_width_last_col_dim }
      \bool_if:NT \l_@@_preamble_bool
        {
          \int_compare:nNnT { \c@jCol } < { \g_@@_static_num_of_col_int }
            { \@@_err_columns_not_used: }
        }
      \@@_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.

```
      \egroup
```

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

```
      \iow_now:Nn \@mainaux { \ExplSyntaxOn }
      \iow_now:Nn \@mainaux { \char_set_catcode_space:n { 32 }  }
      \iow_now:Ne \@mainaux
        {
          \tl_gclear_new:c { g_@@_ \int_use:N \g_@@_env_int _ tl }
          \tl_gset:cn { g_@@_ \int_use:N \g_@@_env_int _ tl }
            { \exp_not:o \g_@@_aux_tl }
        }
      \iow_now:Nn \@mainaux { \ExplSyntaxOff }

      \bool_if:NT \g_@@_footnote_bool { \endsavenotes }
    }
```

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

```
\cs_new_protected:Npn \@@_err_columns_not_used:
  {
    \@@_warning:n { columns~not~used }
    \cs_gset:Npn \@@_err_columns_not_used: { }
  }
```

The following command will be used only once. We have written that command for legibility. 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 1.0. For a `X`-column of weight $x$, the width will be `\l_@@_X_columns_dim` multiplied by $x$.

```
\cs_new_protected:Npn \@@_compute_width_X:
  {
    \tl_gput_right:Ne \g_@@_aux_tl
      {
        \bool_set_true:N \l_@@_X_columns_aux_bool
        \dim_set:Nn \l_@@_X_columns_dim
          {
```

The flag `g_@@_V_of_X_bool` is raised when there is at least in the tabular a column of type `X` using the key `V`. In that case, the width of the `X` column may be considered as correct even though the tabular has not (of course) a width equal to `\l_@@_width_dim`

```
            \bool_lazy_and:nnTF
              { \g_@@_V_of_X_bool }
              { \l_@@_X_columns_aux_bool }
              { \dim_use:N \l_@@_X_columns_dim }
              {
                \dim_compare:nNnTF
                  {
                    \dim_abs:n
                      { \l_@@_width_dim - \box_wd:N \l_@@_the_array_box }
                  }
                  <
                  { 0.001 pt }
                  { \dim_use:N \l_@@_X_columns_dim }
                  {
                    \dim_eval:n
                      {
                        \l_@@_X_columns_dim
                        +
                        \fp_to_dim:n
                          {
                            (
                              \dim_eval:n
                                { \l_@@_width_dim - \box_wd:N \l_@@_the_array_box }
                            )
                            / \fp_use:N \g_@@_total_X_weight_fp
                          }
                      }
                  }
              }
          }
      }
  }
```

## 11 Construction of 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`.

```
\cs_new_protected:Npn \@@_transform_preamble:
  {
    \@@_transform_preamble_i:
    \@@_transform_preamble_ii:
  }
```

```
\cs_new_protected:Npn \@@_transform_preamble_i:
  {
    \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`).

```
    \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.

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

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

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

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

```
    \int_zero:N \l_tmpa_int
    \tl_gclear:N \g_@@_array_preamble_tl
    \str_if_eq:eeTF { \l_@@_vlines_clist } { all }
      {
        \tl_gset:Nn \g_@@_array_preamble_tl
          { ! { \skip_horizontal:N \arrayrulewidth } }
      }
      {
        \clist_if_in:NnT \l_@@_vlines_clist 1
          {
            \tl_gset:Nn \g_@@_array_preamble_tl
              { ! { \skip_horizontal:N \arrayrulewidth } }
          }
      }
```

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

```
    \exp_last_unbraced:No \@@_rec_preamble:n \g_@@_user_preamble_tl \s_stop
    \int_gset_eq:NN \g_@@_static_num_of_col_int \c@jCol

    \@@_replace_columncolor:
  }


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

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

```
     \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
       {
         \tl_if_eq:NNF \g_@@_right_delim_tl \c_@@_dot_tl
           { \bool_gset_true:N \g_@@_delims_bool }
       }
       { \bool_gset_true:N \g_@@_delims_bool }
```

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

```
    \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).

```
    \int_if_zero:nTF { \l_@@_first_col_int }
      { \tl_gput_left:No \g_@@_array_preamble_tl \c_@@_preamble_first_col_tl }
      {
        \bool_if:NF \g_@@_delims_bool
          {
            \bool_if:NF \l_@@_tabular_bool
              {
                \clist_if_empty:NT \l_@@_vlines_clist
                  {
                    \bool_if:NF \l_@@_exterior_arraycolsep_bool
                      { \tl_gput_left:Nn \g_@@_array_preamble_tl { @ { } } }
                  }
              }
          }
      }
    \int_compare:nNnTF { \l_@@_last_col_int } > { -1 }
      { \tl_gput_right:No \g_@@_array_preamble_tl \c_@@_preamble_last_col_tl }
      {
        \bool_if:NF \g_@@_delims_bool
          {
            \bool_if:NF \l_@@_tabular_bool
              {
                \clist_if_empty:NT \l_@@_vlines_clist
                  {
                    \bool_if:NF \l_@@_exterior_arraycolsep_bool
                      { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { } } }
                  }
              }
          }
      }
```

We try to give a good error message when the final user puts more columns than allowed by the preamble of the array. The mechanism consists of an extra column. However, if tagging is in force, that dummy extra column will be tagged (with `<TD>` tags) and that's why we disable that mechanism when tagging is in force.

```
    \tag_if_active:F
      {
```

Moreover, when `{NiceTabular*}` is used, the mechanism can't be used for technical reasons. We test that situation with `\l_@@_tabular_width_dim`.

```
        \dim_compare:nNnT { \l_@@_tabular_width_dim } = { \c_zero_dim }
          {
            \tl_gput_right:Nn \g_@@_array_preamble_tl
              { > { \@@_err_too_many_cols: } l }
          }
      }
  }
```

We have used to add a last column to raise a good error message when the user puts more columns than allowed by its preamble. For technical reasons, it was not possible to do that in `{NiceTabular*}` and that's why we used to control that with the value of `\l_@@_tabular_width_dim`).

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.

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

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.[11]

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

Now, the columns defined by `\newcolumntype` of array.

```
        \cs_if_exist:cTF { NC @ find @ #1 }
          {
            \tl_set_eq:Nc \l_tmpb_tl { NC @ rewrite @ #1 }
            \exp_last_unbraced:No \@@_rec_preamble:n \l_tmpb_tl
          }
          {
```

[11]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`.

```
            \str_if_eq:nnTF { #1 } { S }
              { \@@_fatal:n { unknown~column~type~S } }
              { \@@_fatal:nn { unknown~column~type } { #1 } }
          }
      }
  }
```

For c, l and r

```
\cs_new_protected:Npn \@@_c: #1
  {
    \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
    \tl_gclear:N \g_@@_pre_cell_tl
    \tl_gput_right:Nn \g_@@_array_preamble_tl
      { > \@@_cell_begin: c < \@@_cell_end: }
```

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

```
    \int_gincr:N \c@jCol
    \@@_rec_preamble_after_col:n
  }
\cs_new_protected:Npn \@@_l: #1
  {
    \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
    \tl_gclear:N \g_@@_pre_cell_tl
    \tl_gput_right:Nn \g_@@_array_preamble_tl
      {
        > { \@@_cell_begin: \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_l_tl }
        l
        < \@@_cell_end:
      }
    \int_gincr:N \c@jCol
    \@@_rec_preamble_after_col:n
  }
\cs_new_protected:Npn \@@_r: #1
  {
    \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
    \tl_gclear:N \g_@@_pre_cell_tl
    \tl_gput_right:Nn \g_@@_array_preamble_tl
      {
        > { \@@_cell_begin: \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_r_tl }
        r
        < \@@_cell_end:
      }
    \int_gincr:N \c@jCol
    \@@_rec_preamble_after_col:n
  }
```

For ! and @

```
\cs_new_protected:cpn { @@ _ \token_to_str:N ! : } #1 #2
  {
    \tl_gput_right:Nn \g_@@_array_preamble_tl { #1 { #2 } }
    \@@_rec_preamble:n
  }
\cs_set_eq:cc { @@ _ \token_to_str:N @ : } { @@ _ \token_to_str:N ! : }
```

For |

```
\cs_new_protected:cpn { @@ _ | : } #1
  {
```

\l_tmpa_int is the number of successive occurrences of |

```
    \int_incr:N \l_tmpa_int
    \@@_make_preamble_i_i:n
  }
```

```
\cs_new_protected:Npn \@@_make_preamble_i_i:n #1
  {
```

Here, we can't use `\str_if_eq:eeTF`.

```
    \str_if_eq:nnTF { #1 } { | }
      { \use:c { @@ _ | : } | }
      { \@@_make_preamble_i_ii:nn { } #1 }
  }
```

The following constructions aims to allow cumulative blocks of options between square brackets such as in `|[color=blue][tikz=dashed]`.

```
\cs_new_protected:Npn \@@_make_preamble_i_ii:nn #1 #2
  {
    \str_if_eq:nnTF { #2 } { [ }
      { \@@_make_preamble_i_ii:nw { #1 } [ }
      { \@@_make_preamble_i_iii:nn { #2 } { #1 } }
  }
\cs_new_protected:Npn \@@_make_preamble_i_ii:nw #1 [ #2 ]
  { \@@_make_preamble_i_ii:nn { #1 , #2 } }
```

```
\cs_new_protected:Npn \@@_make_preamble_i_iii:nn #1 #2
  {
    \@@_compute_rule_width:n { multiplicity = \l_tmpa_int , #2 }
    \tl_gput_right:Ne \g_@@_array_preamble_tl
      {
```

Here, the command `\dim_use:N` is mandatory.

```
        \exp_not:N ! { \skip_horizontal:N \dim_use:N \l_@@_rule_width_dim }
      }
    \tl_gput_right:Ne \g_@@_pre_code_after_tl
      {
        \@@_vline:n
          {
            position = \int_eval:n { \c@jCol + 1 } ,
            multiplicity = \int_use:N \l_tmpa_int ,
            total-width = \dim_use:N \l_@@_rule_width_dim ,
            #2
          }
```

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.

```
      }
    \int_zero:N \l_tmpa_int
    \str_if_eq:nnT { #1 } { \s_stop } { \bool_gset_true:N \g_tmpb_bool }
    \@@_rec_preamble:n #1
  }
```

```
\cs_new_protected:cpn { @@ _  > : } #1 #2
  {
    \tl_gput_right:Nn \g_@@_pre_cell_tl { > { #2 } }
    \@@_rec_preamble:n
  }
```

```
\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.

```
\keys_define:nn { nicematrix / p-column }
  {
    r .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_r_str ,
    r .value_forbidden:n = true ,
    c .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_c_str ,
    c .value_forbidden:n = true ,
    l .code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_l_str ,
    l .value_forbidden:n = true ,
    S .code:n = \str_set:Nn \l_@@_hpos_col_str { si } ,
    S .value_forbidden:n = true ,
    p .code:n = \str_set:Nn \l_@@_vpos_col_str { p } ,
    p .value_forbidden:n = true ,
    t .meta:n = p ,
    m .code:n = \str_set:Nn \l_@@_vpos_col_str { m } ,
    m .value_forbidden:n = true ,
    b .code:n = \str_set:Nn \l_@@_vpos_col_str { b } ,
    b .value_forbidden:n = true
  }
```

For `p` but also `b` and `m`.

```
\cs_new_protected:Npn \@@_p: #1
  {
    \str_set:Nn \l_@@_vpos_col_str { #1 }
```

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

```
    \@@_make_preamble_ii_i:n
  }
\cs_new_eq:NN \@@_b: \@@_p:
\cs_new_eq:NN \@@_m: \@@_p:

\cs_new_protected:Npn \@@_make_preamble_ii_i:n #1
  {
    \str_if_eq:nnTF { #1 } { [ }
      { \@@_make_preamble_ii_ii:w [ }
      { \@@_make_preamble_ii_ii:w [ ] { #1 } }
  }

\cs_new_protected:Npn \@@_make_preamble_ii_ii:w [ #1 ]
  { \@@_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.

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

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).

```
    \str_set:Nn \l_@@_hpos_col_str { j }
    \@@_keys_p_column:n { #1 }
```

We apply `setlength` in order to allow a width of column of the form `\widthof{Some words}`. `\widthof` is a command of the package calc (not loaded by nicematrix) which redefines the command `\setlength`. Of course, even if calc is not loaded, the following code will work with the standard version of `\setlength`.

```
    \setlength { \l_tmpa_dim } { #2 }
    \@@_make_preamble_ii_iv:nnn { \dim_use:N \l_tmpa_dim } { minipage } { }
  }

\cs_new_protected:Npn \@@_keys_p_column:n #1
  { \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.

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

Here, `\expanded` would probably be slightly faster than `\use:e`

```
    \use:e
      {
        \@@_make_preamble_ii_vi:nnnnnnnn
          { \str_if_eq:eeTF { \l_@@_vpos_col_str } { p } { t } { b } }
          { #1 }
          {
            \cs_set_eq:NN \rotate \@@_rotate_p_col:
```

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.

```
            \str_if_eq:eeTF { \l_@@_hpos_col_str } { j }
              { \tl_clear:N \exp_not:N \l_@@_hpos_cell_tl }
              {
```

Here, we use `\def` instead of `\tl_set:Nn` for efficiency only.

```
                \def \exp_not:N \l_@@_hpos_cell_tl
                  { \str_lowercase:f { \l_@@_hpos_col_str } }
              }
            \IfPackageLoadedTF { ragged2e }
              {
                \str_case:on \l_@@_hpos_col_str
                  {
```

The following `\exp_not:N` are mandatory.

```
                    c { \exp_not:N \Centering }
                    l { \exp_not:N \RaggedRight }
                    r { \exp_not:N \RaggedLeft }
                  }
              }
              {
                \str_case:on \l_@@_hpos_col_str
                  {
                    c { \exp_not:N \centering }
                    l { \exp_not:N \raggedright }
                    r { \exp_not:N \raggedleft }
                  }
              }
            #3
          }
          { \str_if_eq:eeT { \l_@@_vpos_col_str } { m } \@@_center_cell_box: }
          { \str_if_eq:eeT { \l_@@_hpos_col_str } { si } \siunitx_cell_begin:w }
          { \str_if_eq:eeT { \l_@@_hpos_col_str } { si } \siunitx_cell_end: }
          { #2 }
          {
            \str_case:onF \l_@@_hpos_col_str
              {
                { j } { c }
                { si } { c }
              }
```

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

```
              { \str_lowercase:f \l_@@_hpos_col_str }
          }
      }
```

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

```
    \int_gincr:N \c@jCol
    \@@_rec_preamble_after_col:n
  }
```

`#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 specifier of column which is used *in fine*.

```
\cs_new_protected:Npn \@@_make_preamble_ii_vi:nnnnnnnn #1 #2 #3 #4 #5 #6 #7 #8
  {
    \str_if_eq:eeTF { \l_@@_hpos_col_str } { si }
      {
        \tl_gput_right:Nn \g_@@_array_preamble_tl
          { > \@@_test_if_empty_for_S: }
      }
      {
        \str_if_eq:eeTF { #7 } { varwidth }
          {
            \tl_gput_right:Nn \g_@@_array_preamble_tl
              { > \@@_test_if_empty_varwidth: }
          }
          { \tl_gput_right:Nn \g_@@_array_preamble_tl { > \@@_test_if_empty: } }
      }
    \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
    \tl_gclear:N \g_@@_pre_cell_tl
    \tl_gput_right:Nn \g_@@_array_preamble_tl
      {
        > {
```

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.

```
            \dim_set:Nn \l_@@_col_width_dim { #2 }
            \@@_cell_begin:
```

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

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

The following lines have been taken from array.sty.

```
            \everypar
              {
                \vrule height \box_ht:N \@arstrutbox width \c_zero_dim
                \everypar { }
              }
```

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

```
            #3
```

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

```
            \g_@@_row_style_tl
            \arraybackslash
            #5
          }
        #8
        < {
            #6
```

The following line has been taken from array.sty.

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

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

```
            #4
            \@@_cell_end:
          }
      }
  }
```

The cell always begins with `\ignorespaces` with array and that's why we retrieve that token.

```
\cs_new_protected:Npn \@@_test_if_empty: \ignorespaces
  {
```

We open a special group with `\group_align_safe_begin:`. Thus, when `\peek_meaning:NTF` will read the `&` (when the cell is empty), that lecture won't trigger the end of the cell (since we are in a lower group...). If the end of cell was trigerred, we would have other tokens in the TeX flow (and not `&`).

```
    \group_align_safe_begin:
    \peek_meaning:NTF &
      { \@@_the_cell_is_empty: }
      {
        \peek_meaning:NTF \\
          { \@@_the_cell_is_empty: }
          {
            \peek_meaning:NTF \crcr
              \@@_the_cell_is_empty:
              \group_align_safe_end:
          }
      }
  }
```

A special version of the previous function for the columns of type `V` (of varwidth).

```
\cs_new_protected:Npn \@@_test_if_empty_varwidth: \ignorespaces
  {
    \group_align_safe_begin:
    \peek_meaning:NTF &
      { \@@_the_cell_is_empty_varwidth: }
      {
        \peek_meaning:NTF \\
          { \@@_the_cell_is_empty_varwidth: }
          {
            \peek_meaning:NTF \crcr
              \@@_the_cell_is_empty_varwidth:
              \group_align_safe_end:
          }
      }
  }
```

```
\cs_new_protected:Npn \@@_the_cell_is_empty:
  {
    \group_align_safe_end:
    \tl_gput_right:Nn \g_@@_cell_after_hook_tl
      {
```

Be careful: here, we can't merely use `\bool_gset_true: \g_@@_empty_cell_bool`, in particular because of the columns of type `X`.

```
        \box_set_wd:Nn \l_@@_cell_box \c_zero_dim
```

If all the cells of the column are empty, we still must have a column with the width required by the column of type `p` (or `b`, or `m`).

```
        \skip_horizontal:N \l_@@_col_width_dim
      }
  }
```

```
\cs_new_protected:Npn \@@_the_cell_is_empty_varwidth:
  {
    \group_align_safe_end:
    \tl_gput_right:Nn \g_@@_cell_after_hook_tl
      { \box_set_wd:Nn \l_@@_cell_box \c_zero_dim }
  }
```

```
\cs_new_protected:Npn \@@_test_if_empty_for_S:
  {
    \peek_meaning:NT \__siunitx_table_skip:n
      { \bool_gset_true:N \g_@@_empty_cell_bool }
  }
```

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.

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

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

```
    \tl_gput_right:Nn \g_@@_cell_after_hook_tl
      {
        \dim_compare:nNnT
          { \box_ht:N \l_@@_cell_box }
          >
```

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).

```
          { \box_ht:N \strutbox }
          {
            \hbox_set:Nn \l_@@_cell_box
              {
                \box_move_down:nn
                  {
                    ( \box_ht:N \l_@@_cell_box - \box_ht:N \@arstrutbox
                      + \baselineskip ) / 2
                  }
                  { \box_use:N \l_@@_cell_box }
              }
          }
      }
  }
```

For V (similar to the V of varwidth).

```
\cs_new_protected:Npn \@@_V: #1 #2
  {
    \str_if_eq:nnTF { #2 } { [ }
      { \@@_make_preamble_V_i:w [ }
      { \@@_make_preamble_V_i:w [ ] { #2 } }
  }
\cs_new_protected:Npn \@@_make_preamble_V_i:w [ #1 ]
  { \@@_make_preamble_V_ii:nn { #1 } }
\cs_new_protected:Npn \@@_make_preamble_V_ii:nn #1 #2
  {
    \str_set:Nn \l_@@_vpos_col_str { p }
    \str_set:Nn \l_@@_hpos_col_str { j }
    \@@_keys_p_column:n { #1 }
```

We apply `setlength` in order to allow a width of column of the form `\widthof{Some words}`. `\widthof` is a command of the package calc (not loaded by nicematrix) which redefines the command `\setlength`. Of course, even if calc is not loaded, the following code will work with the standard version of `\setlength`.

```
    \setlength { \l_tmpa_dim } { #2 }
    \IfPackageLoadedTF { varwidth }
      { \@@_make_preamble_ii_iv:nnn { \dim_use:N \l_tmpa_dim } { varwidth } { } }
      {
        \@@_error_or_warning:n { varwidth~not~loaded }
        \@@_make_preamble_ii_iv:nnn { \dim_use:N \l_tmpa_dim } { minipage } { }
      }
  }
```

For w and W

```
\cs_new_protected:Npn \@@_w: { \@@_make_preamble_w:nnnn { } }
\cs_new_protected: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.

```
\cs_new_protected:Npn \@@_make_preamble_w:nnnn #1 #2 #3 #4
  {
    \str_if_eq:nnTF { #3 } { s }
      { \@@_make_preamble_w_i:nnnn { #1 } { #4 } }
      { \@@_make_preamble_w_ii:nnnn { #1 } { #2 } { #3 } { #4 } }
  }
```

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.

```
\cs_new_protected:Npn \@@_make_preamble_w_i:nnnn #1 #2
  {
    \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
    \tl_gclear:N \g_@@_pre_cell_tl
    \tl_gput_right:Nn \g_@@_array_preamble_tl
      {
        > {
```

We use \setlength in order to allow \widthof which is a command of calc (when loaded calc redefines \setlength). Of course, even if calc is not loaded, the following code will work with the standard version of \setlength.

```
            \setlength { \l_@@_col_width_dim } { #2 }
            \@@_cell_begin:
            \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_c_tl
          }
        c
        < {
            \@@_cell_end_for_w_s:
            #1
            \@@_adjust_size_box:
            \box_use_drop:N \l_@@_cell_box
          }
      }
    \int_gincr:N \c@jCol
    \@@_rec_preamble_after_col:n
  }
```

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

```
\cs_new_protected:Npn \@@_make_preamble_w_ii:nnnn #1 #2 #3 #4
  {
    \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
    \tl_gclear:N \g_@@_pre_cell_tl
    \tl_gput_right:Nn \g_@@_array_preamble_tl
      {
        > {
```

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.
We use `\setlength` in order to allow `\widthof` which is a command of calc (when loaded calc redefines `\setlength`). Of course, even if calc is not loaded, the following code will work with the standard version of `\setlength`.

```
            \setlength { \l_@@_col_width_dim } { #4 }
            \hbox_set:Nw \l_@@_cell_box
            \@@_cell_begin:
            \tl_set:Nn \l_@@_hpos_cell_tl { #3 }
          }
        c
        < {
            \@@_cell_end:
            \hbox_set_end:
            #1
            \@@_adjust_size_box:
            \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
          }
      }
```

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

```
    \int_gincr:N \c@jCol
    \@@_rec_preamble_after_col:n
  }
```

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

For `S` (of siunitx).

```
\cs_new_protected:Npn \@@_S: #1 #2
  {
    \str_if_eq:nnTF { #2 } { [ }
      { \@@_make_preamble_S:w [ }
      { \@@_make_preamble_S:w [ ] { #2 } }
  }
\cs_new_protected:Npn \@@_make_preamble_S:w [ #1 ]
  { \@@_make_preamble_S_i:n { #1 } }
\cs_new_protected:Npn \@@_make_preamble_S_i:n #1
  {
    \IfPackageLoadedF { siunitx } { \@@_fatal:n { siunitx~not~loaded } }
    \tl_gput_right:No \g_@@_array_preamble_tl \g_@@_pre_cell_tl
    \tl_gclear:N \g_@@_pre_cell_tl
    \tl_gput_right:Nn \g_@@_array_preamble_tl
      {
        > {
```

In the cells of a column of type `S`, we have to wrap the command `\@@_node_cell:` for the horizontal alignment of the content of the cell (siunitx has done a job but it's without effect since we have to put the content in a box for the PGF/TikZ node and that's why we have to do the job of horizontal alignement once again).

```
            \socket_assign_plug:nn { nicematrix / siunitx-wrap } { active }
            \keys_set:nn { siunitx } { #1 }
            \@@_cell_begin:
            \siunitx_cell_begin:w
          }
        c
        <
          {
            \siunitx_cell_end:
```

We want the value of `\l__siunitx_table_text_bool` available *after* `\@@_cell_end:` because we need it to know how to align our box after the construction of the PGF/TikZ node. That's why we use `\g_@@_cell_after_hook_tl` to reset the correct value of `\l__siunitx_table_text_bool` (of course, if will stay local within the cell of the underlying `\halign`).

```
            \tl_gput_right:Ne \g_@@_cell_after_hook_tl
              {
                \bool_if:NTF \l__siunitx_table_text_bool
                  { \bool_set_true:N }
                  { \bool_set_false:N }
                \l__siunitx_table_text_bool
              }
            \@@_cell_end:
          }
      }
```

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

```
    \int_gincr:N \c@jCol
    \@@_rec_preamble_after_col:n
  }
```

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

```
\cs_new_protected:cpn { @@ _ \token_to_str:N ( : } #1 #2
  {
    \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.

```
    \int_if_zero:nTF { \c@jCol }
      {
        \tl_if_eq:NNTF \g_@@_left_delim_tl \c_@@_dot_tl
          {
```

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

```
            \tl_gset:Nn \g_@@_left_delim_tl { #1 }
            \tl_gset_eq:NN \g_@@_right_delim_tl \c_@@_dot_tl
            \@@_rec_preamble:n #2
          }
          {
            \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
            \@@_make_preamble_iv:nn { #1 } { #2 }
          }
      }
      { \@@_make_preamble_iv:nn { #1 } { #2 } }
  }
\cs_set_eq:cc { @@ _ \token_to_str:N [ : } { @@ _ \token_to_str:N ( : }
\cs_set_eq:cc { @@ _ \token_to_str:N \{ : } { @@ _ \token_to_str:N ( : }
```

```
\cs_new_protected:Npn \@@_make_preamble_iv:nn #1 #2
  {
    \tl_gput_right:Ne \g_@@_pre_code_after_tl
      { \@@_delimiter:nnn #1 { \int_eval:n { \c@jCol + 1 } } \c_true_bool }
    \tl_if_in:nnTF { ( [ \{ ) ] \} \left \right } { #2 }
      {
        \@@_error:nn { delimiter~after~opening } { #2 }
        \@@_rec_preamble:n
      }
      { \@@_rec_preamble:n #2 }
  }
```

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

```
\cs_new_protected: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}).

```
\cs_new_protected:cpn { @@ _ \token_to_str:N ) : } #1 #2
  {
    \bool_if:NT \l_@@_small_bool { \@@_fatal:n { Delimiter~with~small } }
    \tl_if_in:nnTF { ) ] \} } { #2 }
      { \@@_make_preamble_v:nnn #1 #2 }
      {
        \str_if_eq:nnTF { \s_stop } { #2 }
          {
            \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
              { \tl_gset:Nn \g_@@_right_delim_tl { #1 } }
              {
                \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
                \tl_gput_right:Ne \g_@@_pre_code_after_tl
                  { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
                \@@_rec_preamble:n #2
              }
          }
          {
            \tl_if_in:nnT { ( [ \{ \left } { #2 }
              { \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } } }
            \tl_gput_right:Ne \g_@@_pre_code_after_tl
              { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
            \@@_rec_preamble:n #2
          }
      }
  }
\cs_set_eq:cc { @@ _ \token_to_str:N ] : } { @@ _ \token_to_str:N ) : }
\cs_set_eq:cc { @@ _ \token_to_str:N \} : } { @@ _ \token_to_str:N ) : }
\cs_new_protected:Npn \@@_make_preamble_v:nnn #1 #2 #3
  {
    \str_if_eq:nnTF { \s_stop } { #3 }
      {
        \tl_if_eq:NNTF \g_@@_right_delim_tl \c_@@_dot_tl
          {
            \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
            \tl_gput_right:Ne \g_@@_pre_code_after_tl
              { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
            \tl_gset:Nn \g_@@_right_delim_tl { #2 }
          }
          {
            \tl_gput_right:Nn \g_@@_array_preamble_tl { ! { \enskip } }
            \tl_gput_right:Ne \g_@@_pre_code_after_tl
              { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
            \@@_error:nn { double~closing~delimiter } { #2 }
          }
      }
      {
        \tl_gput_right:Ne \g_@@_pre_code_after_tl
          { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
        \@@_error:nn { double~closing~delimiter } { #2 }
        \@@_rec_preamble:n #3
      }
  }

\cs_new_protected:cpn { @@ _ \token_to_str:N \right : } #1
  { \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 `@{...}`.

```
\cs_new_protected:Npn \@@_rec_preamble_after_col:n #1
  {
    \str_if_eq:nnTF { #1 } { < }
      { \@@_rec_preamble_after_col_i:n }
      {
        \str_if_eq:nnTF { #1 } { @ }
          { \@@_rec_preamble_after_col_ii:n }
          {
            \str_if_eq:eeTF { \l_@@_vlines_clist } { all }
              {
                \tl_gput_right:Nn \g_@@_array_preamble_tl
                  { ! { \skip_horizontal:N \arrayrulewidth } }
              }
              {
                \clist_if_in:NeT \l_@@_vlines_clist
                  { \int_eval:n { \c@jCol + 1 } }
                  {
                    \tl_gput_right:Nn \g_@@_array_preamble_tl
                      { ! { \skip_horizontal:N \arrayrulewidth } }
                  }
              }
            \@@_rec_preamble:n { #1 }
          }
      }
  }
```

```
\cs_new_protected:Npn \@@_rec_preamble_after_col_i:n #1
  {
    \tl_gput_right:Nn \g_@@_array_preamble_tl { < { #1 } }
    \@@_rec_preamble_after_col:n
  }
```

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.

```
\cs_new_protected:Npn \@@_rec_preamble_after_col_ii:n #1
  {
    \str_if_eq:eeTF { \l_@@_vlines_clist } { all }
      {
        \tl_gput_right:Nn \g_@@_array_preamble_tl
          { @ { #1 \skip_horizontal:N \arrayrulewidth } }
      }
      {
        \clist_if_in:NeTF \l_@@_vlines_clist { \int_eval:n { \c@jCol + 1 } }
          {
            \tl_gput_right:Nn \g_@@_array_preamble_tl
              { @ { #1 \skip_horizontal:N \arrayrulewidth } }
          }
          { \tl_gput_right:Nn \g_@@_array_preamble_tl { @ { #1 } } }
      }
    \@@_rec_preamble:n
  }
```

```
\cs_new_protected:cpn { @@ _ * : } #1 #2 #3
  {
    \tl_clear:N \l_tmpa_tl
    \int_step_inline:nn { #2 } { \tl_put_right:Nn \l_tmpa_tl { #3 } }
    \exp_last_unbraced:No \@@_rec_preamble:n \l_tmpa_tl
  }
```

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

```
\cs_new_protected: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`.

```
\cs_new_protected:Npn \@@_X: #1 #2
  {
    \str_if_eq:nnTF { #2 } { [ }
      { \@@_make_preamble_X:w [ }
      { \@@_make_preamble_X:w [ ] #2 }
  }
\cs_new_protected:Npn \@@_make_preamble_X:w [ #1 ]
  { \@@_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 `V` and also a key which corresponds to a positive number (1, 2, 0.5, etc.) which is the *weight* of the columns. The following set of keys will be used to retrieve that value and store it in `\l_tmpa_fp`.

```
\keys_define:nn { nicematrix / X-column }
  {
    V .code:n =
      \IfPackageLoadedTF { varwidth }
        {
          \bool_set_true:N \l_@@_V_of_X_bool
          \bool_gset_true:N \g_@@_V_of_X_bool
        }
        { \@@_error_or_warning:n { varwidth~not~loaded~in~X } } ,
    unknown .code:n =
      \regex_if_match:nVTF { \A[0-9]*\.?[0-9]*\Z } \l_keys_key_str
        { \fp_set:Nn \l_tmpa_fp { \l_keys_key_str } }
        { \@@_error_or_warning:n { invalid~weight } }
  }
```

In the following command, `#1` is the list of the options of the specifier `X`.

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

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`).

```
    \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`).

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

We will store in `\l_tmpa_fp` the weight of the column (`\l_tmpa_fp` also appears in `{nicematrix/X-column}` and the error message `invalid~weight`.

```
    \fp_set:Nn \l_tmpa_fp { 1.0 }
    \@@_keys_p_column:n { #1 }
```

The unknown keys have been stored by `\@@_keys_p_column:n` in `\l_tmpa_tl` and we use them right away in the set of keys `nicematrix/X-column` in order to retrieve the potential weight explicitely provided by the final user.

```
    \bool_set_false:N \l_@@_V_of_X_bool
    \keys_set:no { nicematrix / X-column } \l_tmpa_tl
```

Now, the weight of the column is stored in `\l_tmpa_tl`.

```
    \fp_gadd:Nn \g_@@_total_X_weight_fp \l_tmpa_fp
```

We test whether we know the actual 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).

```
    \bool_if:NTF \l_@@_X_columns_aux_bool
      {
        \@@_make_preamble_ii_iv:nnn
```

Of course, the weight of a column depends of its weight (in \l_tmpa_fp).

```
          { \fp_use:N \l_tmpa_fp \l_@@_X_columns_dim }
          { \bool_if:NTF \l_@@_V_of_X_bool { varwidth } { minipage } }
          { \@@_no_update_width: }
      }
```

In the current compilation, we don't known the actual width of the X column. However, you have to construct the cells of that column! By convention, we have decided to compose in a {minipage} of width 5 cm even though we will nullify \l_@@_cell_box after its composition.

```
      {
        \tl_gput_right:Nn \g_@@_array_preamble_tl
          {
            > {
                \@@_cell_begin:
                \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.

```
                \NotEmpty
```

The following code will nullify the box of the cell.

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

```
                \begin { minipage } { 5 cm } \arraybackslash
              }
            c
            < {
                \end { minipage }
                \@@_cell_end:
              }
          }
        \int_gincr:N \c@jCol
        \@@_rec_preamble_after_col:n
      }
  }

\cs_new_protected:Npn \@@_no_update_width:
  {
    \tl_gput_right:Nn \g_@@_cell_after_hook_tl
      { \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing: }
  }
```

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

```
\cs_new_protected:Npn \@@_make_preamble_vlism:n #1
  {
    \seq_gput_right:Ne \g_@@_cols_vlism_seq
      { \int_eval:n { \c@jCol + 1 } }
    \tl_gput_right:Ne \g_@@_array_preamble_tl
      { \exp_not:N ! { \skip_horizontal:N \arrayrulewidth } }
    \@@_rec_preamble:n
  }
```

The token `\s_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.

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

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

```
\cs_new_protected:cpn { @@ _ \token_to_str:N \hline : }
  { \@@_fatal:n { Preamble~forgotten } }
\cs_set_eq:cc { @@ _ \token_to_str:N \Hline : } { @@ _ \token_to_str:N \hline : }
\cs_set_eq:cc { @@ _ \token_to_str:N \toprule : }
  { @@ _ \token_to_str:N \hline : }
\cs_set_eq:cc { @@ _ \token_to_str:N \Block : } { @@ _ \token_to_str:N \hline : }
\cs_set_eq:cc { @@ _ \token_to_str:N \CodeBefore : }
  { @@ _ \token_to_str:N \hline : }
\cs_set_eq:cc { @@ _ \token_to_str:N \RowStyle : }
  { @@ _ \token_to_str:N \hline : }
\cs_set_eq:cc { @@ _ \token_to_str:N \diagbox : }
  { @@ _ \token_to_str:N \hline : }
\cs_set_eq:cc { @@ _ \token_to_str:N & : }
  { @@ _ \token_to_str:N \hline : }
```

## 12 The redefinition of \multicolumn

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

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

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`.

```
    \multispan { #1 }
    \cs_set_eq:NN \@@_update_max_cell_width: \prg_do_nothing:
    \begingroup
    \tbl_update_multicolumn_cell_data:n { #1 }
```

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

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

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

```
    \def \@addamp
      {
        \legacy_if:nTF { @firstamp }
          { \legacy_if_set_false:n { @firstamp } }
          { \@preamerr 5 }
      }
    \exp_args:No \@mkpream \g_@@_preamble_tl
    \@addtopreamble \@empty
    \endgroup
    \UseTaggingSocket { tbl / colspan } { #1 }
```

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

```
    \int_compare:nNnT { #1 } > { \c_one_int }
      {
        \seq_gput_left:Ne \g_@@_multicolumn_cells_seq
          { \int_use:N \c@iRow - \int_eval:n { \c@jCol + 1 } }
        \seq_gput_left:Nn \g_@@_multicolumn_sizes_seq { #1 }
        \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
          {
            {
              \int_if_zero:nTF { \c@jCol }
                { \int_eval:n { \c@iRow + 1 } }
                { \int_use:N \c@iRow }
            }
            { \int_eval:n { \c@jCol + 1 } }
            {
              \int_if_zero:nTF { \c@jCol }
                { \int_eval:n { \c@iRow + 1 } }
                { \int_use:N \c@iRow }
            }
            { \int_eval:n { \c@jCol + #1 } }
```

The last argument is for the name of the block.

```
            { }
          }
      }
```

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

```
    \RenewDocumentCommand { \cellcolor } { O { } m }
      {
        \tl_gput_right:Ne \g_@@_pre_code_before_tl
          {
            \@@_rectanglecolor [ ##1 ]
              { \exp_not:n { ##2 } }
              { \int_use:N \c@iRow - \int_use:N \c@jCol }
              { \int_use:N \c@iRow - \int_eval:n { \c@jCol + #1 } }
          }
        \ignorespaces
      }
```

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

```
    \def \@sharp { #3 }
    \@arstrut
    \@preamble
    \null
```

We add some lines.

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

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`.

```
\cs_new_protected:Npn \@@_make_m_preamble:n #1
  {
    \str_case:nnF { #1 }
      {
        c { \@@_make_m_preamble_i:n #1 }
        l { \@@_make_m_preamble_i:n #1 }
        r { \@@_make_m_preamble_i:n #1 }
        > { \@@_make_m_preamble_ii:nn #1 }
        ! { \@@_make_m_preamble_ii:nn #1 }
        @ { \@@_make_m_preamble_ii:nn #1 }
        | { \@@_make_m_preamble_iii:n #1 }
        p { \@@_make_m_preamble_iv:nnn t #1 }
        m { \@@_make_m_preamble_iv:nnn c #1 }
        b { \@@_make_m_preamble_iv:nnn b #1 }
        w { \@@_make_m_preamble_v:nnnn { } #1 }
        W { \@@_make_m_preamble_v:nnnn { \@@_special_W: } #1 }
        \q_stop { }
      }
      {
        \cs_if_exist:cTF { NC @ find @ #1 }
          {
            \tl_set_eq:Nc \l_tmpa_tl { NC @ rewrite @ #1 }
            \exp_last_unbraced:No \@@_make_m_preamble:n \l_tmpa_tl
          }
          {
            \str_if_eq:nnTF { #1 } { S }
              { \@@_fatal:n { unknown~column~type~S~multicolumn } }
              { \@@_fatal:nn { unknown~column~type~multicolumn } { #1 } }
          }
      }
  }
```

For c, l and r

```
\cs_new_protected:Npn \@@_make_m_preamble_i:n #1
  {
    \tl_gput_right:Nn \g_@@_preamble_tl
      {
        > { \@@_cell_begin: \tl_set:Nn \l_@@_hpos_cell_tl { #1 } }
        #1
        < \@@_cell_end:
      }
```

We test for the presence of a <.

```
    \@@_make_m_preamble_x:n
  }
```

For >, ! and @

```
\cs_new_protected:Npn \@@_make_m_preamble_ii:nn #1 #2
  {
    \tl_gput_right:Nn \g_@@_preamble_tl { #1 { #2 } }
    \@@_make_m_preamble:n
  }
```

For |

```
\cs_new_protected:Npn \@@_make_m_preamble_iii:n #1
  {
    \tl_gput_right:Nn \g_@@_preamble_tl { #1 }
    \@@_make_m_preamble:n
  }
```

For p, m and b

```
\cs_new_protected:Npn \@@_make_m_preamble_iv:nnn #1 #2 #3
  {
    \tl_gput_right:Nn \g_@@_preamble_tl
      {
        > {
            \@@_cell_begin:
```

We use \setlength instead of \dim_set:N to allow a specifier like p{\widthof{Some words}}. widthof is a command provided by calc. Of course, even if calc is not loaded, the following code will work with the standard version of \setlength.

```
            \setlength { \l_tmpa_dim } { #3 }
            \begin { minipage } [ #1 ] { \l_tmpa_dim }
            \mode_leave_vertical:
            \arraybackslash
            \vrule height \box_ht:N \@arstrutbox depth \c_zero_dim width \c_zero_dim
```

```
          }
        c
        < {
            \vrule height \c_zero_dim depth \box_dp:N \@arstrutbox width \c_zero_dim
            \end { minipage }
            \@@_cell_end:
          }
      }
```

We test for the presence of a <.

```
    \@@_make_m_preamble_x:n
  }
```

For w and W

```
\cs_new_protected:Npn \@@_make_m_preamble_v:nnnn #1 #2 #3 #4
  {
    \tl_gput_right:Nn \g_@@_preamble_tl
      {
        > {
            \dim_set:Nn \l_@@_col_width_dim { #4 }
            \hbox_set:Nw \l_@@_cell_box
            \@@_cell_begin:
            \tl_set:Nn \l_@@_hpos_cell_tl { #3 }
          }
        c
        < {
            \@@_cell_end:
            \hbox_set_end:
            \bool_if:NT \g_@@_rotate_bool { \@@_rotate_cell_box: }
            #1
            \@@_adjust_size_box:
            \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
          }
      }
```

We test for the presence of a <.

```
    \@@_make_m_preamble_x:n
  }
```

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

```
\cs_new_protected:Npn \@@_make_m_preamble_x:n #1
  {
    \str_if_eq:nnTF { #1 } { < }
      { \@@_make_m_preamble_ix:n }
      { \@@_make_m_preamble:n { #1 } }
  }
\cs_new_protected:Npn \@@_make_m_preamble_ix:n #1
  {
    \tl_gput_right:Nn \g_@@_preamble_tl { < { #1 } }
    \@@_make_m_preamble_x:n
  }
```

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`).

```
\cs_new_protected:Npn \@@_put_box_in_flow:
  {
    \box_set_ht:Nn \l_tmpa_box { \box_ht:N \l_tmpa_box + \l_tmpa_dim }
    \box_set_dp:Nn \l_tmpa_box { \box_dp:N \l_tmpa_box + \l_tmpb_dim }
    \str_if_eq:eeTF { \l_@@_baseline_tl } { c }
      { \box_use_drop:N \l_tmpa_box }
      { \@@_put_box_in_flow_i: }
  }
```

The command `\@@_put_box_in_flow_i:` is used when the value of `\l_@@_baseline_tl` is different of `c` (the initial value).

```
\cs_new_protected:Npn \@@_put_box_in_flow_i:
  {
    \pgfpicture
      \@@_qpoint:n { row - 1 }
      \dim_gset_eq:NN \g_tmpa_dim \pgf@y
      \@@_qpoint:n { row - \int_eval:n { \c@iRow + 1 } }
      \dim_gadd:Nn \g_tmpa_dim \pgf@y
      \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).

```
      \tl_if_in:NnTF \l_@@_baseline_tl { line- }
        {
          \int_set:Nn \l_tmpa_int
            { \str_range:Nnn \l_@@_baseline_tl { 6 } { -1 } }
          \bool_lazy_or:nnT
            { \int_compare_p:nNn { \l_tmpa_int } < { 1 } }
            { \int_compare_p:nNn { \l_tmpa_int } > { \c@iRow + 1 } }
            {
              \@@_error:n { bad~value~for~baseline-line }
              \int_set_eq:NN \l_tmpa_int \c_one_int
            }
          \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
        }
        {
          \str_if_eq:eeTF { \l_@@_baseline_tl } { t }
            { \int_set_eq:NN \l_tmpa_int \c_one_int }
            {
              \str_if_eq:onTF \l_@@_baseline_tl  { b }
                { \int_set_eq:NN \l_tmpa_int \c@iRow }
                { \int_set:Nn \l_tmpa_int \l_@@_baseline_tl }
            }
          \bool_lazy_or:nnT
            { \int_compare_p:nNn { \l_tmpa_int } < { \l_@@_first_row_int } }
            { \int_compare_p:nNn { \l_tmpa_int } > { \g_@@_row_total_int } }
            {
              \@@_error:n { bad~value~for~baseline }
              \int_set_eq:NN \l_tmpa_int \c_one_int
            }
          \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }
```

We take into account the position of the mathematical axis.

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

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

```
    \endpgfpicture
    \box_move_up:nn \g_tmpa_dim { \box_use_drop:N \l_tmpa_box }
    \box_use_drop:N \l_tmpa_box
  }
```

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).

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

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.

```
    \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
```

```
        {
          \int_compare:nNnT { \c@jCol } > { \c_one_int }
            {
              \box_set_wd:Nn \l_@@_the_array_box
                { \box_wd:N \l_@@_the_array_box - \arraycolsep }
            }
        }
```

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

```
      \begin { minipage } [ t ] { \box_wd:N \l_@@_the_array_box }
      \bool_if:NT \l_@@_caption_above_bool
        {
          \tl_if_empty:NF \l_@@_caption_tl
            {
              \bool_set_false:N \g_@@_caption_finished_bool
              \int_gzero:N \c@tabularnote
              \@@_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).

```
              \int_compare:nNnT { \g_@@_notes_caption_int } > { \c_zero_int }
                {
                  \tl_gput_right:Ne \g_@@_aux_tl
                    {
                      \tl_set:Nn \exp_not:N \l_@@_note_in_caption_tl
                        { \int_use:N \g_@@_notes_caption_int }
                    }
                  \int_gzero:N \g_@@_notes_caption_int
                }
            }
        }
```

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

```
      \hbox
        {
          \box_use_drop:N \l_@@_the_array_box
```

We have to draw the blocks right away 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.

```
          \@@_create_extra_nodes:
          \seq_if_empty:NF \g_@@_blocks_seq { \@@_draw_blocks: }
        }
```

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

```
      \bool_lazy_any:nT
        {
          { ! \seq_if_empty_p:N \g_@@_notes_seq }
          { ! \seq_if_empty_p:N \g_@@_notes_in_caption_seq }
          { ! \tl_if_empty_p:o \g_@@_tabularnote_tl }
        }
        \@@_insert_tabularnotes:
      \cs_set_eq:NN \tabularnote \@@_tabularnote_error:n
      \bool_if:NF \l_@@_caption_above_bool { \@@_insert_caption: }
      \end { minipage }
    }
```

```
\cs_new_protected:Npn \@@_insert_caption:
  {
    \tl_if_empty:NF \l_@@_caption_tl
      {
        \cs_if_exist:NTF \@captype
          { \@@_insert_caption_i: }
          { \@@_error:n { caption~outside~float } }
      }
  }
```

```
\cs_new_protected:Npn \@@_insert_caption_i:
  {
    \group_begin:
```

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

```
      \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.

```
      \IfPackageLoadedT { floatrow } { \cs_set_eq:NN \@makecaption \FR@makecaption }
      \tl_if_empty:NTF \l_@@_short_caption_tl
        { \caption }
        { \caption [ \l_@@_short_caption_tl ] }
        { \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.

```
      \bool_if:NF \g_@@_caption_finished_bool
        {
          \bool_gset_true:N \g_@@_caption_finished_bool
          \int_gset_eq:NN \g_@@_notes_caption_int \c@tabularnote
          \int_gzero:N \c@tabularnote
        }
      \tl_if_empty:NF \l_@@_label_tl { \label { \l_@@_label_tl } }
      \group_end:
  }
\cs_new_protected:Npn \@@_tabularnote_error:n #1
  {
    \@@_error_or_warning:n { tabularnote~below~the~tabular }
    \cs_gset:Npn \@@_tabularnote_error:n ##1 { }
  }
\cs_new_protected:Npn \@@_insert_tabularnotes:
  {
    \seq_gconcat:NNN \g_@@_notes_seq \g_@@_notes_in_caption_seq \g_@@_notes_seq
    \int_set:Nn \c@tabularnote { \seq_count:N \g_@@_notes_seq }
    \skip_vertical:N 0.65ex
```

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

```
    \group_begin:
    \l_@@_notes_code_before_tl
    \tl_if_empty:NF \g_@@_tabularnote_tl
      {
        \g_@@_tabularnote_tl \par
        \tl_gclear:N \g_@@_tabularnote_tl
      }
```

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.

```
    \int_compare:nNnT { \c@tabularnote } > { \c_zero_int }
      {
        \bool_if:NTF \l_@@_notes_para_bool
          {
            \begin { tabularnotes* }
              \seq_map_inline:Nn \g_@@_notes_seq
                { \@@_one_tabularnote:nn ##1 }
              \strut
            \end { tabularnotes* }
```

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

```
            \par
          }
          {
            \tabularnotes
              \seq_map_inline:Nn \g_@@_notes_seq
                { \@@_one_tabularnote:nn ##1 }
              \strut
            \endtabularnotes
          }
      }
    \unskip
    \group_end:
    \bool_if:NT \l_@@_notes_bottomrule_bool
      {
        \IfPackageLoadedTF { booktabs }
          {
```

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

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

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

```
            { \CT@arc@ \hrule height \heavyrulewidth }
          }
          { \@@_error_or_warning:n { bottomrule~without~booktabs } }
      }
    \l_@@_notes_code_after_tl
    \seq_gclear:N \g_@@_notes_seq
    \seq_gclear:N \g_@@_notes_in_caption_seq
    \int_gzero:N \c@tabularnote
  }
```

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.

```
\cs_set_protected:Npn \@@_one_tabularnote:nn #1
  {
    \tl_if_novalue:nTF { #1 }
      { \item }
      { \item [ \@@_notes_label_in_list:n { #1 } ] }
  }
```

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`.

```
\cs_new_protected:Npn \@@_use_arraybox_with_notes_b:
  {
    \pgfpicture
      \@@_qpoint:n { row - 1 }
      \dim_gset_eq:NN \g_tmpa_dim \pgf@y
      \@@_qpoint:n { row - \int_use:N \c@iRow - base }
      \dim_gsub:Nn \g_tmpa_dim \pgf@y
    \endpgfpicture
    \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
    \int_if_zero:nT { \l_@@_first_row_int }
      {
        \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
        \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
      }
    \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
  }
```

Now, the general case.

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

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

```
    \str_if_eq:eeT { \l_@@_baseline_tl } { t }
      { \tl_set:Nn \l_@@_baseline_tl { 1 } }
```

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

```
    \pgfpicture
    \@@_qpoint:n { row - 1 }
    \dim_gset_eq:NN \g_tmpa_dim \pgf@y
    \tl_if_in:NnTF \l_@@_baseline_tl { line- }
      {
        \int_set:Nn \l_tmpa_int
          {
            \str_range:Nnn
              \l_@@_baseline_tl
              { 6 }
              { \tl_count:o \l_@@_baseline_tl }
          }
        \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
      }
      {
        \int_set:Nn \l_tmpa_int \l_@@_baseline_tl
        \bool_lazy_or:nnT
          { \int_compare_p:nNn { \l_tmpa_int } < { \l_@@_first_row_int } }
          { \int_compare_p:nNn { \l_tmpa_int } > { \g_@@_row_total_int } }
          {
            \@@_error:n { bad~value~for~baseline }
            \int_set:Nn \l_tmpa_int 1
          }
        \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }
      }
    \dim_gsub:Nn \g_tmpa_dim \pgf@y
    \endpgfpicture
    \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
    \int_if_zero:nT { \l_@@_first_row_int }
      {
        \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
        \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
      }
    \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
  }
```

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.

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

We will compute the real width of both delimiters used.

```
    \dim_zero_new:N \l_@@_real_left_delim_dim
    \dim_zero_new:N \l_@@_real_right_delim_dim
    \hbox_set:Nn \l_tmpb_box
      {
        \m@th
        $ % $
        \left #1
        \vcenter
          {
            \vbox_to_ht:nn
              { \box_ht_plus_dp:N \l_tmpa_box }
              { }
          }
        \right .
        $ % $
      }
    \dim_set:Nn \l_@@_real_left_delim_dim
      { \box_wd:N \l_tmpb_box - \nulldelimiterspace }
    \hbox_set:Nn \l_tmpb_box
      {
        \m@th
        $ % $
        \left .
        \vbox_to_ht:nn
          { \box_ht_plus_dp:N \l_tmpa_box }
          { }
        \right #2
        $ % $
      }
    \dim_set:Nn \l_@@_real_right_delim_dim
      { \box_wd:N \l_tmpb_box - \nulldelimiterspace }
```

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

```
    \skip_horizontal:n { \l_@@_left_delim_dim - \l_@@_real_left_delim_dim }
    \@@_put_box_in_flow:
    \skip_horizontal:n { \l_@@_right_delim_dim - \l_@@_real_right_delim_dim }
  }
```

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).

```
\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, a standard error will be raised by LaTeX for incorrect nested environments).

```
  {
    \peek_remove_spaces:n
      {
        \peek_meaning:NTF \end
          { \@@_analyze_end:Nn }
          {
            \@@_transform_preamble:
```

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

```
            \@@_array:o \g_@@_array_preamble_tl
          }
      }
  }
  {
    \@@_create_col_nodes:
    \endarray
  }
```

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

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

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`.

```
    \tl_if_empty:nT { #1 }
      { \@@_fatal:n { empty~environment } }
    \tl_if_in:nnT { #1 } { & }
      { \@@_fatal:n { ampersand~in~light-syntax } }
    \tl_if_in:nnT { #1 } { \\ }
      { \@@_fatal:n { double-backslash~in~light-syntax } }
```

Now, you extract the `\CodeAfter` of the body of the environment. Maybe, there is no command `\CodeAfter` in the body. That's why you put a marker `\CodeAfter` after `#1`. If there is yet a `\CodeAfter` in `#1`, this second (or third...) `\CodeAfter` will be caught in the value of `\g_nicematrix_code_after_tl`. That doesn't matter because `\CodeAfter` will be set to *no-op* before the execution of `\g_nicematrix_code_after_tl`.

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

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

```
  }
```

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.

```
  {
    \@@_create_col_nodes:
    \endarray
  }
\cs_new_protected:Npn \@@_light_syntax_i:w #1\CodeAfter #2 \q_stop
  {
    \tl_gput_right:Nn \g_nicematrix_code_after_tl { #2 }
```

The body of the array, which is stored in the argument `#1`, is now split into items (and *not* tokens).

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

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

```
    \tl_set_rescan:Nno \l_@@_end_of_row_tl { } \l_@@_end_of_row_tl
    \bool_if:NTF \l_@@_light_syntax_expanded_bool
      { \seq_set_split:Nee }
      { \seq_set_split:Non }
      \l_@@_rows_seq \l_@@_end_of_row_tl { #1 }
```

We delete the last row if it is empty.

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

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

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

First, we treat the first row.

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

```
    \seq_map_inline:Nn \l_@@_rows_seq
      {
        \tl_build_put_right:Nn \l_@@_new_body_tl { \\ }
        \@@_line_with_light_syntax:n { ##1 }
      }
    \tl_build_end:N \l_@@_new_body_tl

    \int_compare:nNnT { \l_@@_last_col_int } = { -1 }
      {
        \int_set:Nn \l_@@_last_col_int
          { \l_@@_nb_cols_int - 1 + \l_@@_first_col_int }
      }
```

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.

```
    \@@_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).

```
    \@@_array:o \g_@@_array_preamble_tl \l_@@_new_body_tl
  }

\cs_new_protected:Npn \@@_line_with_light_syntax:n #1
  {
    \seq_clear_new:N \l_@@_cells_seq
    \seq_set_split:Nnn \l_@@_cells_seq { ~ } { #1 }
    \int_set:Nn \l_@@_nb_cols_int
      {
        \int_max:nn
          { \l_@@_nb_cols_int }
          { \seq_count:N \l_@@_cells_seq }
      }
    \seq_pop_left:NN \l_@@_cells_seq \l_tmpa_tl
    \tl_build_put_right:No \l_@@_new_body_tl \l_tmpa_tl
    \seq_map_inline:Nn \l_@@_cells_seq
      { \tl_build_put_right:Nn \l_@@_new_body_tl { & ##1 } }
  }
\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`.

```
\cs_new_protected:Npn \@@_analyze_end:Nn #1 #2
  {
    \str_if_eq:eeT { \g_@@_name_env_str } { #2 }
      { \@@_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.

```
    \end { #2 }
  }
```

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`).

```
\cs_new:Npn \@@_create_col_nodes:
  {
    \crcr
    \int_if_zero:nT { \l_@@_first_col_int }
      {
        \omit
        \hbox_overlap_left:n
          {
            \bool_if:NT \l_@@_code_before_bool
              { \pgfsys@markposition { \@@_env: - col - 0 } }
            \pgfpicture
            \pgfrememberpicturepositiononpagetrue
            \pgfcoordinate { \@@_env: - col - 0 } \pgfpointorigin
            \str_if_empty:NF \l_@@_name_str
              { \pgfnodealias { \l_@@_name_str - col - 0 } { \@@_env: - col - 0 } }
            \endpgfpicture
            \skip_horizontal:n { 2 \col@sep + \g_@@_width_first_col_dim }
          }
        &
      }
    \omit
```

The following instruction must be put after the instruction `\omit` since, of course, it is not expandable.

```
    \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`).

```
    \int_if_zero:nTF { \l_@@_first_col_int }
      {
        \@@_mark_position:n { 1 }
        \pgfpicture
        \pgfrememberpicturepositiononpagetrue
        \pgfcoordinate { \@@_env: - col - 1 }
          { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
        \str_if_empty:NF \l_@@_name_str
          { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
        \endpgfpicture
      }
      {
        \bool_if:NT \l_@@_code_before_bool
          {
            \hbox
              {
                \skip_horizontal:n { 0.5 \arrayrulewidth }
                \pgfsys@markposition { \@@_env: - col - 1 }
                \skip_horizontal:n { -0.5 \arrayrulewidth }
              }
          }
        \pgfpicture
        \pgfrememberpicturepositiononpagetrue
        \pgfcoordinate { \@@_env: - col - 1 }
          { \pgfpoint { 0.5 \arrayrulewidth } \c_zero_dim }
        \@@_node_alias:n { 1 }
        \endpgfpicture
      }
```

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.

```
    \skip_gset:Nn \g_tmpa_skip { 0 pt~plus 1 fill }
    \bool_if:NF \l_@@_auto_columns_width_bool
      { \dim_compare:nNnT { \l_@@_columns_width_dim } > { \c_zero_dim } }
      {
        \bool_lazy_and:nnTF
          { \l_@@_auto_columns_width_bool }
          { \bool_not_p:n \l_@@_block_auto_columns_width_bool }
          { \skip_gadd:Nn \g_tmpa_skip \g_@@_max_cell_width_dim }
          { \skip_gadd:Nn \g_tmpa_skip \l_@@_columns_width_dim }
        \skip_gadd:Nn \g_tmpa_skip { 2 \col@sep }
      }
    \skip_horizontal:N \g_tmpa_skip
    \hbox
      {
        \@@_mark_position:n { 2 }
        \pgfpicture
        \pgfrememberpicturepositiononpagetrue
        \pgfcoordinate { \@@_env: - col - 2 }
          { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
        \@@_node_alias:n { 2 }
        \endpgfpicture
      }
```

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.

```
    \int_gset_eq:NN \g_tmpa_int \c_one_int
    \bool_if:NTF \g_@@_last_col_found_bool
      { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 3 } { 0 } } }
      { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 2 } { 0 } } }
      {
        &
        \omit
        \int_gincr:N \g_tmpa_int
```

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

```
        \skip_horizontal:N \g_tmpa_skip
        \@@_mark_position:n { \int_eval:n { \g_tmpa_int + 1 } }
```

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

```
        \pgfpicture
          \pgfrememberpicturepositiononpagetrue
          \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
            { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
          \@@_node_alias:n { \int_eval:n { \g_tmpa_int + 1 } }
        \endpgfpicture
      }
```

If there is only one column (and a potential “last column”), we don’t have to put the following code (there is only one column and we have put the correct code previously).

```
        \bool_lazy_or:nnF
          { \int_compare_p:nNn \g_@@_col_total_int = 1 }
          { \int_compare_p:nNn \g_@@_col_total_int = 2 && \g_@@_last_col_found_bool }
          {
            &
            \omit
            \skip_horizontal:N \g_tmpa_skip
            \int_gincr:N \g_tmpa_int
            \bool_lazy_any:nF
              {
                \g_@@_delims_bool
                \l_@@_tabular_bool
                { ! \clist_if_empty_p:N \l_@@_vlines_clist }
                \l_@@_exterior_arraycolsep_bool
                \l_@@_bar_at_end_of_pream_bool
              }
              { \skip_horizontal:n { - \col@sep } }
            \bool_if:NT \l_@@_code_before_bool
              {
                \hbox
                  {
                    \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.

```
                    \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
                      { \skip_horizontal:n { - \arraycolsep } }
                    \pgfsys@markposition
                      { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
                    \skip_horizontal:n { 0.5 \arrayrulewidth }
                    \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
                      { \skip_horizontal:N \arraycolsep }
                  }
              }
            \pgfpicture
              \pgfrememberpicturepositiononpagetrue
              \pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
                {
                  \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
                    {
                      \pgfpoint
                        { - 0.5 \arrayrulewidth - \arraycolsep }
                        \c_zero_dim
                    }
                    { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
                }
              \@@_node_alias:n { \int_eval:n { \g_tmpa_int + 1 } }
            \endpgfpicture
          }

    \bool_if:NT \g_@@_last_col_found_bool
      {
        \hbox_overlap_right:n
          {
            \skip_horizontal:N \g_@@_width_last_col_dim
            \skip_horizontal:N \col@sep
            \bool_if:NT \l_@@_code_before_bool
              {
                \pgfsys@markposition
                  { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
              }
            \pgfpicture
            \pgfrememberpicturepositiononpagetrue
            \pgfcoordinate
              { \@@_env: - col - \int_eval:n { \g_@@_col_total_int + 1 } }
              \pgfpointorigin
            \@@_node_alias:n { \int_eval:n { \g_@@_col_total_int + 1 } }
            \endpgfpicture
          }
      }
  }
```

```
\cs_new_protected:Npn \@@_mark_position:n #1
  {
    \bool_if:NT \l_@@_code_before_bool
```

```
      {
        \hbox
          {
            \skip_horizontal:n { -0.5 \arrayrulewidth }
            \pgfsys@markposition { \@@_env: - col - #1 }
            \skip_horizontal:n { 0.5 \arrayrulewidth }
          }
      }
  }
\cs_new_protected:Npn \@@_node_alias:n #1
  {
    \str_if_empty:NF \l_@@_name_str
      { \pgfnodealias { \l_@@_name_str - col - #1 } { \@@_env: - col - #1 } }
  }
```

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

```
\tl_const:Nn \c_@@_preamble_first_col_tl
  {
    >
      {
```

At the beginning of the cell, we link `\CodeAfter` to a command which begins with `\\` (whereas the standard version of `\CodeAfter` begins does not).

```
        \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
        \bool_gset_true:N \g_@@_after_col_zero_bool
        \@@_begin_of_row:
        \hbox_set:Nw \l_@@_cell_box
        \@@_math_toggle:
        \@@_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”.

```
        \int_compare:nNnT { \c@iRow } > { \c_zero_int }
          {
            \bool_lazy_or:nnT
              { \int_compare_p:nNn { \l_@@_last_row_int } < { \c_zero_int } }
              { \int_compare_p:nNn { \c@iRow } < { \l_@@_last_row_int } }
              {
                \l_@@_code_for_first_col_tl
                \xglobal \colorlet { nicematrix-first-col } { . }
              }
          }
      }
```

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`.

```
    l
    <
      {
        \@@_math_toggle:
        \hbox_set_end:
        \bool_if:NT \g_@@_rotate_bool { \@@_rotate_cell_box: }
        \@@_adjust_size_box:
        \@@_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.

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

```
        \hbox_overlap_left:n
          {
              \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > { \c_zero_dim }
                { \@@_node_cell: }
                { \box_use_drop:N \l_@@_cell_box }
              \skip_horizontal:N \l_@@_left_delim_dim
              \skip_horizontal:N \l_@@_left_margin_dim
              \skip_horizontal:N \l_@@_extra_left_margin_dim
            }
          \bool_gset_false:N \g_@@_empty_cell_bool
          \skip_horizontal:n { -2 \col@sep }
        }
    }
```

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

```
\tl_const:Nn \c_@@_preamble_last_col_tl
  {
    >
      {
        \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).

```
        \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.

```
        \bool_gset_true:N \g_@@_last_col_found_bool
        \int_gincr:N \c@jCol
        \int_gset_eq:NN \g_@@_col_total_int \c@jCol
        \hbox_set:Nw \l_@@_cell_box
          \@@_math_toggle:
          \@@_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".

```
        \int_compare:nNnT { \c@iRow } > { \c_zero_int }
          {
            \bool_lazy_or:nnT
              { \int_compare_p:nNn { \l_@@_last_row_int } < { \c_zero_int } }
              { \int_compare_p:nNn { \c@iRow } < { \l_@@_last_row_int } }
              {
                \l_@@_code_for_last_col_tl
                \xglobal \colorlet { nicematrix-last-col } { . }
              }
          }
      }
    l
    <
      {
        \@@_math_toggle:
        \hbox_set_end:
        \bool_if:NT \g_@@_rotate_bool { \@@_rotate_cell_box: }
        \@@_adjust_size_box:
        \@@_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.

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

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

```
        \hbox_overlap_right:n
          {
            \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > { \c_zero_dim }
              {
                \skip_horizontal:N \l_@@_right_delim_dim
                \skip_horizontal:N \l_@@_right_margin_dim
                \skip_horizontal:N \l_@@_extra_right_margin_dim
                \@@_node_cell:
              }
          }
        \bool_gset_false:N \g_@@_empty_cell_bool
      }
  }
```

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

```
\NewDocumentEnvironment { NiceArray } { }
  {
    \bool_gset_false:N \g_@@_delims_bool
    \str_if_empty:NT \g_@@_name_env_str
      { \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).

```
    \NiceArrayWithDelims . .
  }
  { \endNiceArrayWithDelims }
```

We create the variants of the environment {NiceArrayWithDelims}.

```
\cs_new_protected:Npn \@@_def_env:NNN #1 #2 #3
  {
    \NewDocumentEnvironment { #1 NiceArray } { }
      {
        \bool_gset_true:N \g_@@_delims_bool
        \str_if_empty:NT \g_@@_name_env_str
          { \str_gset:Nn \g_@@_name_env_str { #1 NiceArray } }
        \@@_test_if_math_mode:
        \NiceArrayWithDelims #2 #3
      }
      { \endNiceArrayWithDelims }
  }
\@@_def_env:NNN p (     )
\@@_def_env:NNN b [     ]
\@@_def_env:NNN B \{    \}
\@@_def_env:NNN v \vert \vert
\@@_def_env:NNN V \Vert \Vert
```

## 13 The environment {NiceMatrix} and its variants

```
\cs_new_protected:Npn \@@_begin_of_NiceMatrix:nn #1 #2
  {
    \bool_set_false:N \l_@@_preamble_bool
    \tl_clear:N \l_tmpa_tl
    \bool_if:NT \l_@@_NiceMatrix_without_vlines_bool
      { \tl_set:Nn \l_tmpa_tl { @ { } } }
    \tl_put_right:Nn \l_tmpa_tl
      {
        *
          {
            \int_case:nnF \l_@@_last_col_int
              {
                { -2 } { \c@MaxMatrixCols }
                { -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).

```
              }
              { \int_eval:n { \l_@@_last_col_int - 1 } }
          }
          { #2 }
      }
    \tl_set:Nn \l_tmpb_tl { \use:c { #1 NiceArray } }
    \exp_args:No \l_tmpb_tl \l_tmpa_tl
  }
\cs_generate_variant:Nn \@@_begin_of_NiceMatrix:nn { n o }
\clist_map_inline:nn { p , b , B , v , V }
  {
    \NewDocumentEnvironment { #1 NiceMatrix } { ! O { } }
      {
        \bool_gset_true:N \g_@@_delims_bool
        \str_gset:Nn \g_@@_name_env_str { #1 NiceMatrix }
        \int_if_zero:nT { \l_@@_last_col_int }
          {
            \bool_set_true:N \l_@@_last_col_without_value_bool
            \int_set:Nn \l_@@_last_col_int { -1 }
          }
        \keys_set:nn { nicematrix / NiceMatrix } { ##1 }
        \@@_begin_of_NiceMatrix:no { #1 } { \l_@@_columns_type_tl }
      }
      { \use:c { end #1 NiceArray } }
  }
```

We define also an environment {NiceMatrix}

```
\NewDocumentEnvironment { NiceMatrix } { ! O { } }
  {
    \str_gset:Nn \g_@@_name_env_str { NiceMatrix }
    \int_if_zero:nT { \l_@@_last_col_int }
      {
        \bool_set_true:N \l_@@_last_col_without_value_bool
        \int_set:Nn \l_@@_last_col_int { -1 }
      }
    \keys_set:nn { nicematrix / NiceMatrix } { #1 }
    \bool_lazy_or:nnT
      { \clist_if_empty_p:N \l_@@_vlines_clist }
      { \l_@@_except_borders_bool }
      { \bool_set_true:N \l_@@_NiceMatrix_without_vlines_bool }
    \@@_begin_of_NiceMatrix:no { } { \l_@@_columns_type_tl }
  }
  { \endNiceArray }
```

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

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

## 14 {NiceTabular}, {NiceTabularX} and {NiceTabular*}

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

If the dimension \l_@@_width_dim is equal to 0 pt, that means that it has not been set by a previous use of \NiceMatrixOptions.

```
    \dim_compare:nNnT { \l_@@_width_dim } = { \c_zero_dim }
      { \dim_set_eq:NN \l_@@_width_dim \linewidth }
    \str_gset:Nn \g_@@_name_env_str { NiceTabular }
    \keys_set:nn { nicematrix / NiceTabular } { #1 , #3 }
    \tl_if_empty:NF \l_@@_short_caption_tl
```

```
      {
        \tl_if_empty:NT \l_@@_caption_tl
          {
            \@@_error_or_warning:n { short-caption~without~caption }
            \tl_set_eq:NN \l_@@_caption_tl \l_@@_short_caption_tl
          }
      }
    \tl_if_empty:NF \l_@@_label_tl
      {
        \tl_if_empty:NT \l_@@_caption_tl
          { \@@_error_or_warning:n { label~without~caption } }
      }
    \NewDocumentEnvironment { TabularNote } { b }
      {
        \bool_if:NTF \l_@@_in_code_after_bool
          { \@@_error_or_warning:n { TabularNote~in~CodeAfter } }
          {
            \tl_if_empty:NF \g_@@_tabularnote_tl
              { \tl_gput_right:Nn \g_@@_tabularnote_tl { \par } }
            \tl_gput_right:Nn \g_@@_tabularnote_tl { ##1 }
          }
      }
      { }
    \@@_settings_for_tabular:
    \NiceArray { #2 }
  }
  { \endNiceArray }
\cs_new_protected:Npn \@@_settings_for_tabular:
  {
    \bool_set_true:N \l_@@_tabular_bool
    \cs_set_eq:NN \@@_math_toggle: \prg_do_nothing:
    \cs_set_eq:NN \@@_tuning_not_tabular_begin: \prg_do_nothing:
    \cs_set_eq:NN \@@_tuning_not_tabular_end: \prg_do_nothing:
  }

\NewDocumentEnvironment { NiceTabularX } { m O { } m ! O { } }
  {
    \str_gset:Nn \g_@@_name_env_str { NiceTabularX }
    \dim_set:Nn \l_@@_width_dim { #1 }
    \keys_set:nn { nicematrix / NiceTabular } { #2 , #4 }
    \@@_settings_for_tabular:
    \NiceArray { #3 }
  }
  {
    \endNiceArray
    \fp_compare:nNnT { \g_@@_total_X_weight_fp } = { \c_zero_fp }
      { \@@_error:n { NiceTabularX~without~X } }
  }

\NewDocumentEnvironment { NiceTabular* } { m O { } m ! O { } }
  {
    \str_gset:Nn \g_@@_name_env_str { NiceTabular* }
    \dim_set:Nn \l_@@_tabular_width_dim { #1 }
    \keys_set:nn { nicematrix / NiceTabular } { #2 , #4 }
    \@@_settings_for_tabular:
    \NiceArray { #3 }
  }
  { \endNiceArray }
```

## 15 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`).

```
\cs_new_protected:Npn \@@_deal_with_rounded_corners:
  {
    \bool_lazy_all:nT
      {
        { \dim_compare_p:nNn { \l_@@_tab_rounded_corners_dim } > { \c_zero_dim } }
        { \l_@@_hvlines_bool }
        { ! \g_@@_delims_bool }
        { ! \l_@@_except_borders_bool }
      }
      {
        \bool_set_true:N \l_@@_except_borders_bool
        \clist_if_empty:NF \l_@@_corners_clist
          { \@@_error:n { hvlines,~rounded-corners~and~corners } }
        \tl_gput_right:Nn \g_@@_pre_code_after_tl
          {
            \@@_stroke_block:nnn
              {
                rounded-corners = \dim_use:N \l_@@_tab_rounded_corners_dim ,
                draw = \l_@@_rules_color_tl
              }
              { 1-1 }
              { \int_use:N \c@iRow - \int_use:N \c@jCol }
          }
      }
  }
```

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

There was a `\hook_gput_code:nnn { env / tabular / begin } { nicematrix }` in the command `\@@_pre_array_after_CodeBefore:` 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.

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

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

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

```
    \bool_if:NT \l_@@_last_row_without_value_bool
      { \int_set_eq:NN \l_@@_last_row_int \g_@@_row_total_int }
    \tl_gput_right:Ne \g_@@_aux_tl
      {
        \seq_gset_from_clist:Nn \exp_not:N \g_@@_size_seq
          {
            \int_use:N \l_@@_first_row_int ,
            \int_use:N \c@iRow ,
            \int_use:N \g_@@_row_total_int ,
            \int_use:N \l_@@_first_col_int ,
            \int_use:N \c@jCol ,
            \int_use:N \g_@@_col_total_int
          }
      }
    \clist_if_empty:NF \g_@@_cbic_clist { \@@_cbic: }
```

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`).

```
    \seq_if_empty:NF \g_@@_pos_of_blocks_seq
      {
        \tl_gput_right:Ne \g_@@_aux_tl
          {
            \seq_gset_from_clist:Nn \exp_not:N \g_@@_pos_of_blocks_seq
              { \seq_use:Nn \g_@@_pos_of_blocks_seq { , } }
          }
      }
    \seq_if_empty:NF \g_@@_multicolumn_cells_seq
      {
        \tl_gput_right:Ne \g_@@_aux_tl
          {
            \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_cells_seq
              { \seq_use:Nn \g_@@_multicolumn_cells_seq { , } }
            \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_sizes_seq
              { \seq_use:Nn \g_@@_multicolumn_sizes_seq { , } }
          }
      }
```

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

```
    \@@_create_diag_nodes:
```

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

```
    \pgfpicture
    \@@_create_aliases_last:
    \str_if_empty:NF \l_@@_name_str { \@@_create_alias_nodes: }
    \endpgfpicture
```

By default, the diagonal lines will be parallelized[12]. 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.

```
    \bool_if:NT \l_@@_parallelize_diags_bool
      {
        \int_gzero:N \g_@@_ddots_int
        \int_gzero:N \g_@@_iddots_int
```

The dimensions `\g_@@_delta_x_one_dim` and `\g_@@_delta_y_one_dim` will contain the $\Delta_x$ and $\Delta_y$ 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 $\Delta_x$ and $\Delta_y$ of the first `\Iddots` diagonal.

```
        \dim_gzero:N \g_@@_delta_x_one_dim
        \dim_gzero:N \g_@@_delta_y_one_dim
        \dim_gzero:N \g_@@_delta_x_two_dim
```

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

```
      \dim_gzero:N \g_@@_delta_y_two_dim
    }
  \bool_set_false:N \l_@@_initial_open_bool
  \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.

```
  \bool_if:NT \l_@@_small_bool { \@@_tuning_key_small_for_dots: }
```

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

```
  \@@_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_clist` which will contain all the cells which are empty (and not in a block) considered in the corners of the array.

```
  \clist_if_empty:NF \l_@@_corners_clist
    {
      \bool_if:NTF \l_@@_no_cell_nodes_bool
        { \@@_error:n { corners~with~no-cell-nodes } }
        { \@@_compute_corners: }
    }
```

By design, we have computed the corners before the adjonction of `\g_@@_future_pos_of_blocks_seq` is used by `\EmptyRow` and `\EmptyColumn` in the `\CodeBefore`.

```
 \seq_gconcat:NNN \g_@@_pos_of_blocks_seq
   \g_@@_pos_of_blocks_seq
   \g_@@_future_pos_of_blocks_seq
 \seq_gclear:N \g_@@_future_pos_of_blocks_seq
```

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

```
  \@@_adjust_pos_of_blocks_seq:
  \@@_deal_with_rounded_corners:
  \clist_if_empty:NF \l_@@_hlines_clist { \@@_draw_hlines: }
  \clist_if_empty:NF \l_@@_vlines_clist { \@@_draw_vlines: }
```

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

```
  \IfPackageLoadedT { tikz }
    {
      \tikzset
        {
          every~picture / .style =
            {
              overlay ,
              remember~picture ,
              name~prefix = \@@_env: -
            }
        }
    }
  \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign:
  \cs_set_eq:NN \SubMatrix \@@_SubMatrix
  \cs_set_eq:NN \UnderBrace \@@_UnderBrace
  \cs_set_eq:NN \OverBrace \@@_OverBrace
  \cs_set_eq:NN \ShowCellNames \@@_ShowCellNames
  \cs_set_eq:NN \TikzEveryCell \@@_TikzEveryCell
  \cs_set_eq:NN \line \@@_line
```

The LaTeX-style boolean \ifmeasuring@ is used by amsmath during the phase of measure in environments such as {align}, etc.

```
    \legacy_if:nF { measuring@ } { \g_@@_pre_code_after_tl }
    \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 \CodeAfter to be *no-op* now.

```
    \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).

```
    \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).

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

```
    \bool_set_true:N \l_@@_in_code_after_bool
    \exp_last_unbraced:No \@@_CodeAfter_keys: \g_nicematrix_code_after_tl
    \scan_stop:
    \tl_gclear:N \g_nicematrix_code_after_tl
    \group_end:
```

\g_@@_pre_code_before_tl is for instructions in the cells of the array such as \rowcolor and \cellcolor. These instructions will be written on the aux file to be added to the code-before in the next run.

```
    \seq_if_empty:NF \g_@@_rowlistcolors_seq { \@@_clear_rowlistcolors_seq: }
    \tl_if_empty:NF \g_@@_pre_code_before_tl
      {
        \tl_gput_right:Ne \g_@@_aux_tl
          {
            \tl_gset:Nn \exp_not:N \g_@@_pre_code_before_tl
              { \exp_not:o \g_@@_pre_code_before_tl }
          }
        \tl_gclear:N \g_@@_pre_code_before_tl
      }
    \tl_if_empty:NF \g_nicematrix_code_before_tl
      {
        \tl_gput_right:Ne \g_@@_aux_tl
          {
            \tl_gset:Nn \exp_not:N \g_@@_code_before_tl
              { \exp_not:o \g_nicematrix_code_before_tl }
          }
        \tl_gclear:N \g_nicematrix_code_before_tl
      }

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

The command \CT@arc@ contains the instruction of color for the rules of the array[13]. 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.

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

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

```
\cs_new_protected:Npn \@@_tuning_key_small_for_dots:
  {
    \dim_set:Nn \l_@@_xdots_radius_dim { 0.7 \l_@@_xdots_radius_dim }
    \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.

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

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`.

```
\NewDocumentCommand \@@_CodeAfter_keys: { O { } }
  { \keys_set:nn { nicematrix / CodeAfter } { #1 } }
```

```
\cs_new_protected:Npn \@@_create_alias_nodes:
  {
    \int_step_inline:nn { \c@iRow }
      {
        \pgfnodealias
          { \l_@@_name_str - ##1 - last }
          { \@@_env: - ##1 - \int_use:N \c@jCol }
      }
    \int_step_inline:nn { \c@jCol }
      {
        \pgfnodealias
          { \l_@@_name_str - last - ##1 }
          { \@@_env: - \int_use:N \c@iRow - ##1 }
      }
    \pgfnodealias
      { \l_@@_name_str - last - last }
      { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
  }
```

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).

```
\cs_new_protected:Npn \@@_adjust_pos_of_blocks_seq:
  {
    \seq_gset_map_e:NNn \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq
      { \@@_adjust_pos_of_blocks_seq_i:nnnnn ##1 }
  }
```

The following command must *not* be protected.

```
\cs_new:Npn \@@_adjust_pos_of_blocks_seq_i:nnnnn #1 #2 #3 #4 #5
  {
    { #1 }
    { #2 }
    {
      \int_compare:nNnTF { #3 } > { 98 }
        { \int_use:N \c@iRow }
        { #3 }
    }
    {
      \int_compare:nNnTF { #4 } > { 98 }
        { \int_use:N \c@jCol }
        { #4 }
    }
    { #5 }
  }
```

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).

```
\hook_gput_code:nnn { begindocument } { . }
  {
    \cs_new_protected:Npe \@@_draw_dotted_lines:
      {
        \c_@@_pgfortikzpicture_tl
        \@@_draw_dotted_lines_i:
        \c_@@_endpgfortikzpicture_tl
      }
  }
```

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

```
\cs_new_protected:Npn \@@_draw_dotted_lines_i:
  {
    \pgfrememberpicturepositiononpagetrue
    \pgf@relevantforpicturesizefalse
    \g_@@_HVdotsfor_lines_tl
    \g_@@_Vdots_lines_tl
    \g_@@_Ddots_lines_tl
    \g_@@_Iddots_lines_tl
    \g_@@_Cdots_lines_tl
    \g_@@_Ldots_lines_tl
  }
```

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

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

```
\pgfdeclareshape { @@_diag_node }
  {
    \savedanchor { \five }
      {
        \dim_gset_eq:NN \pgf@x \l_tmpa_dim
        \dim_gset_eq:NN \pgf@y \l_tmpb_dim
      }
    \anchor { 5 } { \five }
    \anchor { center } { \pgfpointorigin }
    \anchor { 1 }  { \five \pgf@x = 0.2 \pgf@x \pgf@y = 0.2 \pgf@y }
    \anchor { 2 }  { \five \pgf@x = 0.4 \pgf@x \pgf@y = 0.4 \pgf@y }
    \anchor { 25 } { \five \pgf@x = 0.5 \pgf@x \pgf@y = 0.5 \pgf@y }
    \anchor { 3 }  { \five \pgf@x = 0.6 \pgf@x \pgf@y = 0.6 \pgf@y }
    \anchor { 4 }  { \five \pgf@x = 0.8 \pgf@x \pgf@y = 0.8 \pgf@y }
    \anchor { 6 }  { \five \pgf@x = 1.2 \pgf@x \pgf@y = 1.2 \pgf@y }
    \anchor { 7 }  { \five \pgf@x = 1.4 \pgf@x \pgf@y = 1.4 \pgf@y }
    \anchor { 75 } { \five \pgf@x = 1.5 \pgf@x \pgf@y = 1.5 \pgf@y }
    \anchor { 8 }  { \five \pgf@x = 1.6 \pgf@x \pgf@y = 1.6 \pgf@y }
    \anchor { 9 }  { \five \pgf@x = 1.8 \pgf@x \pgf@y = 1.8 \pgf@y }
  }
```

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).

```
\cs_new_protected:Npn \@@_create_diag_nodes:
  {
    \pgfpicture
    \pgfrememberpicturepositiononpagetrue
    \int_step_inline:nn { \int_max:nn { \c@iRow } { \c@jCol } }
      {
        \@@_qpoint:n { col - \int_min:nn { ##1 } { \c@jCol + 1 } }
        \dim_set_eq:NN \l_tmpa_dim \pgf@x
        \@@_qpoint:n { row - \int_min:nn { ##1 } { \c@iRow + 1 } }
        \dim_set_eq:NN \l_tmpb_dim \pgf@y
        \@@_qpoint:n { col - \int_min:nn { ##1 + 1 } { \c@jCol + 1 } }
        \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
        \@@_qpoint:n { row - \int_min:nn { ##1 + 1 } { \c@iRow + 1 } }
        \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
        \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.

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

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

```
    \int_set:Nn \l_tmpa_int { \int_max:nn { \c@iRow } { \c@jCol } + 1 }
    \@@_qpoint:n { row - \int_min:nn { \l_tmpa_int } { \c@iRow + 1 } }
    \dim_set_eq:NN \l_tmpa_dim \pgf@y
    \@@_qpoint:n { col - \int_min:nn { \l_tmpa_int } { \c@jCol + 1 } }
    \pgfcoordinate
      { \@@_env: - \int_use:N \l_tmpa_int } { \pgfpoint \pgf@x \l_tmpa_dim }
    \pgfnodealias
      { \@@_env: - last }
      { \@@_env: - \int_eval:n { \int_max:nn { \c@iRow } { \c@jCol } + 1 } }
    \str_if_empty:NF \l_@@_name_str
      {
        \pgfnodealias
          { \l_@@_name_str - \int_use:N \l_tmpa_int }
          { \@@_env: - \int_use:N \l_tmpa_int }
        \pgfnodealias
          { \l_@@_name_str - last }
          { \@@_env: - last }
      }
    \endpgfpicture
  }
```

## 16 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.

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

The command `\@@_find_extremities: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.

We provide first a version in the L3 syntax, and, then a version slightly more efficient.

```
\cs_new_protected:Npn \@@_find_extremities:nnnn #1 #2 #3 #4
  {
    \cs_set:cpn { @@ _ dotted _ #1 - #2 } { }
    \int_set:Nn \l_@@_initial_i_int { #1 }
    \int_set:Nn \l_@@_initial_j_int { #2 }
    \int_set:Nn \l_@@_final_i_int { #1 }
    \int_set:Nn \l_@@_final_j_int { #2 }
    \bool_set_false:N \l_@@_stop_loop_bool
    \bool_do_until:Nn \l_@@_stop_loop_bool
      {
        \int_add:Nn \l_@@_final_i_int { #3 }
        \int_add:Nn \l_@@_final_j_int { #4 }
        \bool_set_false:N \l_@@_final_open_bool
        \int_compare:nNnTF { \l_@@_final_i_int } > { \l_@@_row_max_int }
          {
            \int_compare:nNnTF { #3 }  = { 1 }
              { \bool_set_true:N \l_@@_final_open_bool }
              {
                \int_compare:nNnT { \l_@@_final_j_int } > { \l_@@_col_max_int }
                  { \bool_set_true:N \l_@@_final_open_bool }
              }
          }
          {
            \int_compare:nNnTF \l_@@_final_j_int < \l_@@_col_min_int
              {
                \int_compare:nNnT { #4 } = { -1 }
                  { \bool_set_true:N \l_@@_final_open_bool }
              }
              {
                \int_compare:nNnT { \l_@@_final_j_int } > { \l_@@_col_max_int }
                  {
                    \int_compare:nNnT { #4 } = { 1 }
                      { \bool_set_true:N \l_@@_final_open_bool }
                  }
              }
          }
        \bool_if:NTF \l_@@_final_open_bool
          {
            \int_sub:Nn \l_@@_final_i_int { #3 }
            \int_sub:Nn \l_@@_final_j_int { #4 }
            \bool_set_true:N \l_@@_stop_loop_bool
          }
          {
            \cs_if_exist:cTF
              {
                @@ _ dotted _
                \int_use:N \l_@@_final_i_int -
                \int_use:N \l_@@_final_j_int
              }
              {
                \int_sub:Nn \l_@@_final_i_int { #3 }
                \int_sub:Nn \l_@@_final_j_int { #4 }
                \bool_set_true:N \l_@@_final_open_bool
                \bool_set_true:N \l_@@_stop_loop_bool
              }
              {
                \cs_if_exist:cTF
                  {
                    pgf @ sh @ ns @ \@@_env:
                    - \int_use:N \l_@@_final_i_int
                    - \int_use:N \l_@@_final_j_int
                  }
                  { \bool_set_true:N \l_@@_stop_loop_bool }
                  {
                    \cs_set_nopar:cpn
                      {
                        @@ _ dotted _
                        \int_use:N \l_@@_final_i_int -
                        \int_use:N \l_@@_final_j_int
                      }
                      { }
                  }
              }
          }
       }
    \bool_set_false:N \l_@@_stop_loop_bool
    \int_set:Nn \l_tmpa_int { \l_@@_col_min_int - 1 }
    \bool_do_until:Nn \l_@@_stop_loop_bool
      {
        \int_sub:Nn \l_@@_initial_i_int { #3 }
        \int_sub:Nn \l_@@_initial_j_int { #4 }
        \bool_set_false:N \l_@@_initial_open_bool
        \int_compare:nNnTF { \l_@@_initial_i_int } < { \l_@@_row_min_int }
          {
            \int_compare:nNnTF { #3} = { 1 }
              { \bool_set_true:N \l_@@_initial_open_bool }
              {
                \int_compare:nNnT { \l_@@_initial_j_int } = { \l_tmpa_int }
                  { \bool_set_true:N \l_@@_initial_open_bool }
              }
          }
          {
            \int_compare:nNnTF { \l_@@_initial_j_int } < { \l_@@_col_min_int }
              {
                \int_compare:nNnT { #4 } = { 1 }
                  { \bool_set_true:N \l_@@_initial_open_bool }
              }
              {
                \int_compare:nNnT { \l_@@_initial_j_int } > { \l_@@_col_max_int }
                  {
                    \int_compare:nNnT { #4 } = { -1 }
                      { \bool_set_true:N \l_@@_initial_open_bool }
                  }
              }
          }
```

```
        \bool_if:NTF \l_@@_initial_open_bool
          {
            \int_add:Nn \l_@@_initial_i_int { #3 }
            \int_add:Nn \l_@@_initial_j_int { #4 }
            \bool_set_true:N \l_@@_stop_loop_bool
          }
          {
            \cs_if_exist:cTF
              {
                @@ _ dotted _
                \int_use:N \l_@@_initial_i_int -
                \int_use:N \l_@@_initial_j_int
              }
              {
                \int_add:Nn \l_@@_initial_i_int { #3 }
                \int_add:Nn \l_@@_initial_j_int { #4 }
                \bool_set_true:N \l_@@_initial_open_bool
                \bool_set_true:N \l_@@_stop_loop_bool
              }
              {
                \cs_if_exist:cTF
                  {
                    pgf @ sh @ ns @ \@@_env:
                    - \int_use:N \l_@@_initial_i_int
                    - \int_use:N \l_@@_initial_j_int
                  }
                  { \bool_set_true:N \l_@@_stop_loop_bool }
                  {
                    \cs_set_nopar:cpn
                      {
                        @@ _ dotted _
                        \int_use:N \l_@@_initial_i_int -
                        \int_use:N \l_@@_initial_j_int
                      }
                      { }
                  }
              }
          }
      }
    \seq_gput_right:Ne \g_@@_pos_of_xdots_seq
      {
        { \int_use:N \l_@@_initial_i_int }
        { \int_min:nn { \l_@@_initial_j_int } { \l_@@_final_j_int } }
        { \int_use:N \l_@@_final_i_int }
        { \int_max:nn { \l_@@_initial_j_int } { \l_@@_final_j_int } }
        { }
      }
  }
```

The following version is slightly more efficient.

```
\cs_new_protected:Npn \@@_find_extremities:nnnn #1 #2 #3 #4
  {
```

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

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

Initialization of variables.

```
    \l_@@_initial_i_int = #1
     \l_@@_initial_j_int = #2
     \l_@@_final_i_int = #1
     \l_@@_final_j_int = #2
```

We will do two loops: one when determining the initial cell and the other when determining the final cell. The boolean `\l_@@_stop_loop_bool` will be used to control these loops. In the first loop, we search the “final” extremity of the line.

```
    \let \l_@@_stop_loop_bool \c_false_bool
    \bool_do_until:Nn \l_@@_stop_loop_bool
      {
```

We test if we are still in the matrix.

```
        \advance \l_@@_final_i_int by #3
        \advance \l_@@_final_j_int by #4
        \let \l_@@_final_open_bool \c_false_bool
        \if_int_compare:w \l_@@_final_i_int > \l_@@_row_max_int
          \if_int_compare:w #3  = \c_one_int
            \let \l_@@_final_open_bool \c_true_bool
          \else:
            \if_int_compare:w \l_@@_final_j_int > \l_@@_col_max_int
               \let \l_@@_final_open_bool \c_true_bool
            \fi:
          \fi:
        \else:
          \if_int_compare:w \l_@@_final_j_int < \l_@@_col_min_int
             \if_int_compare:w #4 = -1
                \let \l_@@_final_open_bool \c_true_bool
             \fi:
          \else:
             \if_int_compare:w \l_@@_final_j_int > \l_@@_col_max_int
                \if_int_compare:w #4 = \c_one_int
                   \let \l_@@_final_open_bool \c_true_bool
                \fi:
             \fi:
          \fi:
        \fi:
        \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.

```
          {
```

We do a step backwards.

```
            \advance \l_@@_final_i_int by - #3
            \advance \l_@@_final_j_int by - #4
            \let \l_@@_stop_loop_bool \c_true_bool
          }
```

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`.

```
          {
            \cs_if_exist:cTF
              {
                @@ _ dotted _
                \int_use:N \l_@@_final_i_int -
                \int_use:N \l_@@_final_j_int
              }
              {
                \advance \l_@@_final_i_int by - #3
                \advance \l_@@_final_j_int by - #4
                \let \l_@@_final_open_bool \c_true_bool
                \let \l_@@_stop_loop_bool \c_true_bool
              }
              {
                \cs_if_exist:cTF
                  {
                    pgf @ sh @ ns @ \@@_env:
                    - \int_use:N \l_@@_final_i_int
                    - \int_use:N \l_@@_final_j_int
                  }
```

```
                    { \let \l_@@_stop_loop_bool \c_true_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.

```
                    {
                      \cs_set_nopar:cpn
                        {
                          @@ _ dotted _
                          \int_use:N \l_@@_final_i_int -
                          \int_use:N \l_@@_final_j_int
                        }
                        { }
                    }
                }
            }
        }
```

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

```
      \let \l_@@_stop_loop_bool \c_false_bool
```

The following line of code is only for efficiency in the following loop.

```
      \l_tmpa_int = \l_@@_col_min_int
      \advance \l_tmpa_int by -1

      \bool_do_until:Nn \l_@@_stop_loop_bool
        {
          \advance \l_@@_initial_i_int by - #3
          \advance \l_@@_initial_j_int by - #4
          \let \l_@@_initial_open_bool \c_false_bool
```

We test if we are still in the matrix. Since this is the core of the loop, we **optimize** the code by using a TeX-style of conditionals.

```
          \if_int_compare:w \l_@@_initial_i_int < \l_@@_row_min_int
            \if_int_compare:w #3 = \c_one_int
              \let \l_@@_initial_open_bool \c_true_bool
            \else:
```

`\l_tmpa_int` contains `\l_@@_col_min_int - 1` (only for efficiency).

```
              \if_int_compare:w \l_@@_initial_j_int = \l_tmpa_int
                \let \l_@@_initial_open_bool \c_true_bool
              \fi:
            \fi:
          \else:
            \if_int_compare:w \l_@@_initial_j_int < \l_@@_col_min_int
              \if_int_compare:w #4 = \c_one_int
                \let \l_@@_initial_open_bool \c_true_bool
              \fi:
            \else:
              \if_int_compare:w \l_@@_initial_j_int > \l_@@_col_max_int
                \if_int_compare:w #4 = -1
                  \let \l_@@_initial_open_bool \c_true_bool
                \fi:
              \fi:
            \fi:
          \fi:

          \bool_if:NTF \l_@@_initial_open_bool
            {
              \advance \l_@@_initial_i_int by #3
              \advance \l_@@_initial_j_int by #4
              \let \l_@@_stop_loop_bool \c_true_bool
            }
            {
              \cs_if_exist:cTF
                {
                  @@ _ dotted _
                  \int_use:N \l_@@_initial_i_int -
                  \int_use:N \l_@@_initial_j_int
                }
                {
                  \advance \l_@@_initial_i_int by #3
                  \advance \l_@@_initial_j_int by #4
                  \let \l_@@_initial_open_bool \c_true_bool
                  \let \l_@@_stop_loop_bool \c_true_bool
                }
                {
                  \cs_if_exist:cTF
                    {
                      pgf @ sh @ ns @ \@@_env:
                      - \int_use:N \l_@@_initial_i_int
                      - \int_use:N \l_@@_initial_j_int
                    }
                    { \let \l_@@_stop_loop_bool \c_true_bool }
                    {
                      \cs_set_nopar:cpn
                        {
                          @@ _ dotted _
                          \int_use:N \l_@@_initial_i_int -
                          \int_use:N \l_@@_initial_j_int
                        }
                        { }
                    }
                }
            }
        }
```

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.

```
    \seq_gput_right:Ne \g_@@_pos_of_xdots_seq
      {
        { \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`.

```
        { \int_min:nn { \l_@@_initial_j_int } { \l_@@_final_j_int } }
        { \int_use:N \l_@@_final_i_int }
        { \int_max:nn { \l_@@_initial_j_int } { \l_@@_final_j_int } }
        { }
      }
  }
```

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 whether the extremities are closed or open) but before the analysis 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.

```
\cs_new_protected:Npn \@@_open_shorten:
  {
    \bool_if:NT \l_@@_initial_open_bool
      { \dim_zero:N \l_@@_xdots_shorten_start_dim }
    \bool_if:NT \l_@@_final_open_bool
      { \dim_zero:N \l_@@_xdots_shorten_end_dim }
  }
```

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

```
\cs_new_protected:Npn \@@_adjust_to_submatrix:nn #1 #2
  {
    \int_set_eq:NN \l_@@_row_min_int \c_one_int
    \int_set_eq:NN \l_@@_col_min_int \c_one_int
    \int_set_eq:NN \l_@@_row_max_int \c@iRow
    \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`.

```
    \seq_if_empty:NF \g_@@_submatrix_seq
      {
        \seq_map_inline:Nn \g_@@_submatrix_seq
          { \@@_adjust_to_submatrix:nnnnnn { #1 } { #2 } ##1 }
      }
  }
```

`#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.
Here is the programmation of that command with the the standard syntax of L3.

```
\cs_new_protected:Npn \@@_adjust_to_submatrix:nnnnnn #1 #2 #3 #4 #5 #6
  {
    \bool_if:nT
      {
        \int_compare_p:n { #3 <= #1 <= #5 }
        &&
        \int_compare_p:n { #4 <= #2 <= #6 }
      }
      {
        \int_set:Nn \l_@@_row_min_int { \int_max:nn \l_@@_row_min_int { #3 } }
        \int_set:Nn \l_@@_col_min_int { \int_max:nn \l_@@_col_min_int { #4 } }
        \int_set:Nn \l_@@_row_max_int { \int_min:nn \l_@@_row_max_int { #5 } }
        \int_set:Nn \l_@@_col_max_int { \int_min:nn \l_@@_col_max_int { #6 } }
      }
  }
```

However, for efficiency, we will use the following version.

```
\cs_new_protected:Npn \@@_adjust_to_submatrix:nnnnnn #1 #2 #3 #4 #5 #6
  {
    \if_int_compare:w #3 > #1
    \else:
      \if_int_compare:w #1 > #5
      \else:
        \if_int_compare:w #4 > #2
        \else:
          \if_int_compare:w #2 > #6
          \else:
            \if_int_compare:w \l_@@_row_min_int < #3 \l_@@_row_min_int = #3 \fi:
            \if_int_compare:w \l_@@_col_min_int < #4 \l_@@_col_min_int = #4 \fi:
            \if_int_compare:w \l_@@_row_max_int > #5 \l_@@_row_max_int = #5 \fi:
            \if_int_compare:w \l_@@_col_max_int > #6 \l_@@_col_max_int = #6 \fi:
          \fi:
        \fi:
      \fi:
    \fi:
  }
```

```
\cs_new_protected:Npn \@@_set_initial_coords:
  {
    \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
    \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
  }
\cs_new_protected:Npn \@@_set_final_coords:
  {
    \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
    \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
  }
\cs_new_protected:Npn \@@_set_initial_coords_from_anchor:n #1
  {
    \pgfpointanchor
      {
        \@@_env:
        - \int_use:N \l_@@_initial_i_int
        - \int_use:N \l_@@_initial_j_int
      }
      { #1 }
    \@@_set_initial_coords:
  }
\cs_new_protected:Npn \@@_set_final_coords_from_anchor:n #1
  {
    \pgfpointanchor
      {
        \@@_env:
        - \int_use:N \l_@@_final_i_int
        - \int_use:N \l_@@_final_j_int
      }
      { #1 }
    \@@_set_final_coords:
  }
```

```
\cs_new_protected:Npn \@@_open_x_initial_dim:
  {
    \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
    \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int }
      {
        \cs_if_exist:cT
          { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
          {
            \pgfpointanchor
              { \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
              { west }
            \dim_set:Nn \l_@@_x_initial_dim
              { \dim_min:nn { \l_@@_x_initial_dim } { \pgf@x } }
          }
      }
```

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

```
    \dim_compare:nNnT { \l_@@_x_initial_dim } = { \c_max_dim }
      {
        \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
        \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
        \dim_add:Nn \l_@@_x_initial_dim \col@sep
      }
  }
```

```
\cs_new_protected:Npn \@@_open_x_final_dim:
  {
    \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
    \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int }
      {
        \cs_if_exist:cT
          { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
```

```
            {
              \pgfpointanchor
                { \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
                { east }
              \dim_compare:nNnT { \pgf@x } > { \l_@@_x_final_dim }
                { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
            }
        }
```

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

```
      \dim_compare:nNnT { \l_@@_x_final_dim } = { - \c_max_dim }
        {
          \@@_qpoint:n { col - \int_eval:n { \l_@@_final_j_int + 1 } }
          \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
          \dim_sub:Nn \l_@@_x_final_dim \col@sep
        }
    }
```

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.

```
\cs_new_protected:Npn \@@_draw_Ldots:nnn #1 #2 #3
  {
    \@@_adjust_to_submatrix:nn { #1 } { #2 }
    \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
      {
        \@@_find_extremities: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.

```
        \bool_if:NT \g_@@_aux_found_bool
          {
            \group_begin:
              \@@_open_shorten:
              \int_if_zero:nTF { #1 }
                { \color { nicematrix-first-row } }
                {
```

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.

```
                  \int_compare:nNnT { #1 } = { \l_@@_last_row_int }
                    { \color { nicematrix-last-row } }
                }
              \keys_set:nn { nicematrix / xdots } { #3 }
              \@@_color:o \l_@@_xdots_color_tl
              \@@_actually_draw_Ldots:
            \group_end:
          }
      }
  }
```

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.

```
\cs_new_protected:Npn \@@_actually_draw_Ldots:
  {
    \bool_if:NTF \l_@@_initial_open_bool
      {
        \@@_open_x_initial_dim:
        \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
        \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
      }
      { \@@_set_initial_coords_from_anchor:n { base~east } }
    \bool_if:NTF \l_@@_final_open_bool
      {
        \@@_open_x_final_dim:
        \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
        \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
      }
      { \@@_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.

```
    \bool_lazy_all:nTF
      {
        \l_@@_initial_open_bool
        \l_@@_final_open_bool
        { \int_compare_p:nNn { \l_@@_initial_i_int } = { \l_@@_last_row_int } }
      }
      {
        \dim_add:Nn \l_@@_y_initial_dim \c_@@_shift_Ldots_last_row_dim
        \dim_add:Nn \l_@@_y_final_dim \c_@@_shift_Ldots_last_row_dim
      }
```

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 (?).

```
     {
       \dim_add:Nn \l_@@_y_initial_dim \l_@@_xdots_radius_dim
       \dim_add:Nn \l_@@_y_final_dim \l_@@_xdots_radius_dim
     }
    \@@_draw_line:
  }
```

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.

```
\cs_new_protected:Npn \@@_draw_Cdots:nnn #1 #2 #3
  {
    \@@_adjust_to_submatrix:nn { #1 } { #2 }
    \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
      {
        \@@_find_extremities: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.

```
        \bool_if:NT \g_@@_aux_found_bool
          {
            \group_begin:
              \@@_open_shorten:
              \int_if_zero:nTF { #1 }
                { \color { nicematrix-first-row } }
                {
```

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.

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

```
                    { \color { nicematrix-last-row } }
                  }
                \keys_set:nn { nicematrix / xdots } { #3 }
                \@@_color:o \l_@@_xdots_color_tl
                \@@_actually_draw_Cdots:
              \group_end:
            }
        }
    }
```

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`.

```
\cs_new_protected:Npn \@@_actually_draw_Cdots:
  {
    \bool_if:NTF \l_@@_initial_open_bool
      { \@@_open_x_initial_dim: }
      { \@@_set_initial_coords_from_anchor:n { mid~east } }
    \bool_if:NTF \l_@@_final_open_bool
      { \@@_open_x_final_dim: }
      { \@@_set_final_coords_from_anchor:n { mid~west } }
    \bool_lazy_and:nnTF
      { \l_@@_initial_open_bool }
      { \l_@@_final_open_bool }
      {
        \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int }
        \dim_set_eq:NN \l_tmpa_dim \pgf@y
        \@@_qpoint:n { row - \int_eval:n { \l_@@_initial_i_int + 1 } }
        \dim_set:Nn \l_@@_y_initial_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
        \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
      }
      {
        \bool_if:NT \l_@@_initial_open_bool
          { \dim_set_eq:NN \l_@@_y_initial_dim \l_@@_y_final_dim }
        \bool_if:NT \l_@@_final_open_bool
          { \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim }
      }
    \@@_draw_line:
  }
```

```
\cs_new_protected:Npn \@@_open_y_initial_dim:
  {
    \dim_set:Nn \l_@@_y_initial_dim { - \c_max_dim }
    \int_step_inline:nnn { \l_@@_first_col_int } { \g_@@_col_total_int }
      {
        \cs_if_exist:cT
          { pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
          {
            \pgfpointanchor
              { \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
              { north }
            \dim_compare:nNnT { \pgf@y } > { \l_@@_y_initial_dim }
              { \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y }
          }
      }
    \dim_compare:nNnT { \l_@@_y_initial_dim } = { - \c_max_dim }
      {
        \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
        \dim_set:Nn \l_@@_y_initial_dim
          {
            \fp_to_dim:n
              {
                \pgf@y
                + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
              }
          }
      }
  }
\cs_new_protected:Npn \@@_open_y_final_dim:
  {
    \dim_set_eq:NN \l_@@_y_final_dim \c_max_dim
    \int_step_inline:nnn { \l_@@_first_col_int } { \g_@@_col_total_int }
      {
        \cs_if_exist:cT
          { pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
          {
            \pgfpointanchor
              { \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
              { south }
            \dim_compare:nNnT { \pgf@y } < { \l_@@_y_final_dim }
              { \dim_set_eq:NN \l_@@_y_final_dim \pgf@y }
          }
      }
    \dim_compare:nNnT { \l_@@_y_final_dim } = { \c_max_dim }
      {
        \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
        \dim_set:Nn \l_@@_y_final_dim
          { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
      }
  }
```

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.

```
\cs_new_protected:Npn \@@_draw_Vdots:nnn #1 #2 #3
  {
    \@@_adjust_to_submatrix:nn { #1 } { #2 }
    \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
      {
        \@@_find_extremities: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.

```
        \bool_if:NT \g_@@_aux_found_bool
          {
            \group_begin:
              \@@_open_shorten:
              \int_if_zero:nTF { #2 }
                { \color { nicematrix-first-col } }
                {
                  \int_compare:nNnT { #2 } = { \l_@@_last_col_int }
                    { \color { nicematrix-last-col } }
                }
              \keys_set:nn { nicematrix / xdots } { #3 }
              \@@_color:o \l_@@_xdots_color_tl
              \bool_if:NTF \l_@@_Vbrace_bool
                { \@@_actually_draw_Vbrace: }
                { \@@_actually_draw_Vdots: }
              \group_end:
          }
```

```
    }
  }
```

The following function is used by regular calls of `\Vdots` or `\Vdotsfor` but not by `\Vbrace`. 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`.

```
\cs_new_protected:Npn \@@_actually_draw_Vdots:
  {
    \bool_lazy_and:nnTF { \l_@@_initial_open_bool } { \l_@@_final_open_bool }
      { \@@_actually_draw_Vdots_i: }
      { \@@_actually_draw_Vdots_ii: }
    \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
    \@@_draw_line:
  }
```

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

```
\cs_new_protected:Npn \@@_actually_draw_Vdots_i:
  {
    \@@_open_y_initial_dim:
    \@@_open_y_final_dim:
    \int_if_zero:nTF { \l_@@_initial_j_int }
```

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

```
      {
        \@@_qpoint:n { col - 1 }
        \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
        \dim_sub:Nn \l_@@_x_initial_dim
          { \@@_colsep: + \l_@@_left_margin_dim + \l_@@_extra_left_margin_dim }
      }
      {
        \bool_lazy_and:nnTF
          { \int_compare_p:nNn { \l_@@_last_col_int } > { -2 } }
          {
            \int_compare_p:nNn
              { \l_@@_initial_j_int } = { \g_@@_col_total_int }
          }
```

We have a dotted line open on both sides and which is in the “last column”.

```
          {
            \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
            \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
            \dim_add:Nn \l_@@_x_initial_dim
               { \@@_colsep: + \l_@@_right_margin_dim + \l_@@_extra_right_margin_dim }
          }
```

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

```
          {
            \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
            \dim_set_eq:NN \l_tmpa_dim \pgf@x
            \@@_qpoint:n { col - \int_eval:n { \l_@@_initial_j_int + 1 } }
            \dim_set:Nn \l_@@_x_initial_dim { ( \pgf@x + \l_tmpa_dim ) / 2 }
          }
      }
  }
```

The command `\@@_draw_line:` is in `\@@_actually_draw_Vdots:`

Now, the dotted line is *not* open on both sides (maybe open on only one side).
The main task is to determine the *x*-value of the dotted line to draw.
The boolean `\l_tmpa_bool` will indicate whether the column is of type `l` or may be considered as if.

```
\cs_new_protected:Npn \@@_actually_draw_Vdots_ii:
  {
    \bool_set_false:N \l_tmpa_bool
    \bool_if:NF \l_@@_initial_open_bool
      {
        \bool_if:NF \l_@@_final_open_bool
          {
            \@@_set_initial_coords_from_anchor:n { south~west }
            \@@_set_final_coords_from_anchor:n { north~west }
            \bool_set:Nn \l_tmpa_bool
              {
                \dim_compare_p:nNn
                  { \l_@@_x_initial_dim } = { \l_@@_x_final_dim }
              }
          }
      }
```

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

```
    \bool_if:NTF \l_@@_initial_open_bool
      {
        \@@_open_y_initial_dim:
        \@@_set_final_coords_from_anchor:n { north }
        \dim_set_eq:NN \l_@@_x_initial_dim \l_@@_x_final_dim
      }
      {
        \@@_set_initial_coords_from_anchor:n { south }
        \bool_if:NTF \l_@@_final_open_bool
          { \@@_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.

```
          {
            \@@_set_final_coords_from_anchor:n { north }
            \dim_compare:nNnF { \l_@@_x_initial_dim } = { \l_@@_x_final_dim }
              {
                \dim_set:Nn \l_@@_x_initial_dim
                  {
                    \bool_if:NTF \l_tmpa_bool { \dim_min:nn } { \dim_max:nn }
                      \l_@@_x_initial_dim \l_@@_x_final_dim
                  }
              }
          }
      }
  }
```

The following function is used by `\Vbrace` but not by regular uses of `\Vdots` or `\Vdotsfor`.
The command `\@@_actually_draw_Vbrace:` 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`.

```
\cs_new_protected:Npn \@@_actually_draw_Vbrace:
  {
    \bool_if:NTF \l_@@_initial_open_bool
      { \@@_open_y_initial_dim: }
      { \@@_set_initial_coords_from_anchor:n { south } }
    \bool_if:NTF \l_@@_final_open_bool
      { \@@_open_y_final_dim: }
      { \@@_set_final_coords_from_anchor:n { north } }
```

Now, we have the correct values for the $y$-values of both extremities of the brace. We have to compute the $x$-value (there is only one $x$-value since, of course, the brace is vertical).

If we are in the first (exterior) column, the brace must be drawn right flush.

```
    \int_if_zero:nTF { \l_@@_initial_j_int }
      {
        \@@_qpoint:n { col - 1 }
        \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
        \dim_sub:Nn \l_@@_x_initial_dim
          { \@@_colsep: + \l_@@_left_margin_dim + \l_@@_extra_left_margin_dim }
      }
```

Elsewhere, the brace must be drawn left flush.

```
      {
        \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
        \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
        \dim_add:Nn \l_@@_x_initial_dim
          { \@@_colsep: + \l_@@_right_margin_dim + \l_@@_extra_right_margin_dim }
      }
```

We draw a vertical rule and that's why, of course, both $x$-values are equal.

```
    \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
    \@@_draw_line:
  }
```

```
\cs_new:Npn \@@_colsep:
  { \bool_if:NTF \l_@@_tabular_bool { \tabcolsep } { \arraycolsep } }
```

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.

```
\cs_new_protected:Npn \@@_draw_Ddots:nnn #1 #2 #3
  {
    \@@_adjust_to_submatrix:nn { #1 } { #2 }
    \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
      {
        \@@_find_extremities: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.

```
        \bool_if:NT \g_@@_aux_found_bool
          {
            \group_begin:
              \@@_open_shorten:
              \keys_set:nn { nicematrix / xdots } { #3 }
              \@@_color:o \l_@@_xdots_color_tl
              \@@_actually_draw_Ddots:
            \group_end:
          }
      }
  }
```

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`.

```
\cs_new_protected:Npn \@@_actually_draw_Ddots:
  {
    \bool_if:NTF \l_@@_initial_open_bool
      {
        \@@_open_y_initial_dim:
        \@@_open_x_initial_dim:
      }
      { \@@_set_initial_coords_from_anchor:n { south~east } }
    \bool_if:NTF \l_@@_final_open_bool
      {
        \@@_open_x_final_dim:
        \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
      }
      { \@@_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.

```
    \bool_if:NT \l_@@_parallelize_diags_bool
      {
        \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).

```
        \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 $\Delta_x$ and the $\Delta_y$ of the line because these values will be used to draw the others diagonal lines parallels to the first one.

```
          {
            \dim_gset:Nn \g_@@_delta_x_one_dim
              { \l_@@_x_final_dim - \l_@@_x_initial_dim }
            \dim_gset:Nn \g_@@_delta_y_one_dim
              { \l_@@_y_final_dim - \l_@@_y_initial_dim }
          }
```

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`.

```
          {
            \dim_compare:nNnF { \g_@@_delta_x_one_dim } = { \c_zero_dim }
              {
                \dim_set:Nn \l_@@_y_final_dim
                  {
                    \l_@@_y_initial_dim +
                    ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
                    \dim_ratio:nn \g_@@_delta_y_one_dim \g_@@_delta_x_one_dim
                  }
              }
          }
      }
    \@@_draw_line:
  }
```

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.

```
\cs_new_protected:Npn \@@_draw_Iddots:nnn #1 #2 #3
  {
    \@@_adjust_to_submatrix:nn { #1 } { #2 }
    \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
      {
        \@@_find_extremities: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.

```
        \bool_if:NT \g_@@_aux_found_bool
          {
            \group_begin:
              \@@_open_shorten:
              \keys_set:nn { nicematrix / xdots } { #3 }
              \@@_color:o \l_@@_xdots_color_tl
              \@@_actually_draw_Iddots:
            \group_end:
          }
      }
  }
```

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`.

```
\cs_new_protected:Npn \@@_actually_draw_Iddots:
  {
    \bool_if:NTF \l_@@_initial_open_bool
      {
        \@@_open_y_initial_dim:
        \@@_open_x_initial_dim:
      }
      { \@@_set_initial_coords_from_anchor:n { south~west } }
    \bool_if:NTF \l_@@_final_open_bool
      {
        \@@_open_y_final_dim:
        \@@_open_x_final_dim:
      }
      { \@@_set_final_coords_from_anchor:n { north~east } }
    \bool_if:NT \l_@@_parallelize_diags_bool
      {
        \int_gincr:N \g_@@_iddots_int
        \int_compare:nNnTF { \g_@@_iddots_int } = { \c_one_int }
          {
            \dim_gset:Nn \g_@@_delta_x_two_dim
              { \l_@@_x_final_dim - \l_@@_x_initial_dim }
            \dim_gset:Nn \g_@@_delta_y_two_dim
              { \l_@@_y_final_dim - \l_@@_y_initial_dim }
          }
          {
            \dim_compare:nNnF { \g_@@_delta_x_two_dim } = { \c_zero_dim }
              {
                \dim_set:Nn \l_@@_y_final_dim
                  {
                    \l_@@_y_initial_dim +
                    ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
                    \dim_ratio:nn \g_@@_delta_y_two_dim \g_@@_delta_x_two_dim
                  }
              }
          }
      }
    \@@_draw_line:
}
```

## 17 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`

```
\cs_new_protected:Npn \@@_draw_line:
  {
    \pgfrememberpicturepositiononpagetrue
    \pgf@relevantforpicturesizefalse
    \bool_lazy_or:nnTF
      { \tl_if_eq_p:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl }
      { \l_@@_dotted_bool }
      { \@@_draw_standard_dotted_line: }
      { \@@_draw_unstandard_dotted_line: }
  }
```

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.

```
\cs_new_protected:Npn \@@_draw_unstandard_dotted_line:
  {
    \begin { scope }
    \@@_draw_unstandard_dotted_line:o
      { \l_@@_xdots_line_style_tl , \l_@@_xdots_color_tl }
  }
```

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.

```
\cs_new_protected:Npn \@@_draw_unstandard_dotted_line:n #1
  {
    \@@_draw_unstandard_dotted_line:nooo
      { #1 }
      \l_@@_xdots_up_tl
      \l_@@_xdots_down_tl
      \l_@@_xdots_middle_tl
  }
\cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:n { o }
```

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

```
\hook_gput_code:nnn { begindocument } { . }
  {
    \IfPackageLoadedT { tikz }
      {
        \tikzset
          {
            @@_node_above / .style = { sloped , above } ,
            @@_node_below / .style = { sloped , below } ,
            @@_node_middle / .style =
              {
                sloped ,
                inner~sep = \c_@@_innersep_middle_dim
              }
          }
      }
  }
```

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

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 $\ell$ of the line to draw. We use the floating point reals of the L3 programming layer to compute this length.

```
    \dim_zero_new:N \l_@@_l_dim
    \dim_set:Nn \l_@@_l_dim
      {
        \fp_to_dim:n
          {
            sqrt
             (
               ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
                  +
               ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
             )
          }
      }
```

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.

```
    \dim_compare:nNnT { \l_@@_l_dim } < { \c_@@_max_l_dim }
      {
        \dim_compare:nNnT { \l_@@_l_dim }  > { 1 pt }
          \@@_draw_unstandard_dotted_line_i:
      }
```

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

```
    \bool_if:NT \l_@@_xdots_h_labels_bool
      {
        \tikzset
          {
            @@_node_above / .style = { auto = left } ,
            @@_node_below / .style = { auto = right } ,
            @@_node_middle / .style = { inner~sep = \c_@@_innersep_middle_dim }
          }
      }
    \tl_if_empty:nF { #4 }
      { \tikzset { @@_node_middle / .append~style = { fill = white } } }
    \dim_zero:N \l_tmpa_dim
    \dim_zero:N \l_tmpb_dim
    \tl_if_eq:NNTF \l_@@_xdots_line_style_tl \c_@@_brace_tl
      {
```

We test whether the brace is vertical or horizontal.

```
        \dim_compare:nNnTF \l_@@_x_final_dim = \l_@@_x_initial_dim
          { \dim_set_eq:NN \l_tmpa_dim \l_@@_brace_shift_dim }
          { \dim_set_eq:NN \l_tmpb_dim \l_@@_brace_shift_dim }
      }
      {
        \tl_if_eq:NNT \l_@@_xdots_line_style_tl \c_@@_mirrored_brace_tl
          {
            \dim_compare:nNnTF \l_@@_x_final_dim = \l_@@_x_initial_dim
              { \dim_set:Nn \l_tmpa_dim { - \l_@@_brace_shift_dim } }
              { \dim_set:Nn \l_tmpb_dim { - \l_@@_brace_shift_dim } }
          }
      }
    \use:e
      {
        \exp_not:N \begin { scope }
          [ shift = {(\dim_use:N \l_tmpa_dim,\dim_use:N \l_tmpb_dim)} ]
      }
    \draw
      [ #1 ]
          ( \l_@@_x_initial_dim , \l_@@_y_initial_dim )
       -- node [ @@_node_middle] { $ \scriptstyle #4 $ }
          node [ @@_node_below ] { $ \scriptstyle #3 $ }
          node [ @@_node_above ] { $ \scriptstyle #2 $ }
          ( \l_@@_x_final_dim , \l_@@_y_final_dim ) ;
    \end { scope }
    \end { scope }
  }
\cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:nnnn { n o o o }

\cs_new_protected:Npn \@@_draw_unstandard_dotted_line_i:
  {
    \dim_set:Nn \l_tmpa_dim
      {
        \l_@@_x_initial_dim
        + ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
        * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
      }
    \dim_set:Nn \l_tmpb_dim
      {
        \l_@@_y_initial_dim
        + ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
        * \dim_ratio:nn \l_@@_xdots_shorten_start_dim \l_@@_l_dim
      }
    \dim_set:Nn \l_@@_tmpc_dim
      {
        \l_@@_x_final_dim
        - ( \l_@@_x_final_dim - \l_@@_x_initial_dim )
        * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
      }
    \dim_set:Nn \l_@@_tmpd_dim
      {
        \l_@@_y_final_dim
        - ( \l_@@_y_final_dim - \l_@@_y_initial_dim )
        * \dim_ratio:nn \l_@@_xdots_shorten_end_dim \l_@@_l_dim
      }
    \dim_set_eq:NN \l_@@_x_initial_dim \l_tmpa_dim
    \dim_set_eq:NN \l_@@_y_initial_dim \l_tmpb_dim
    \dim_set_eq:NN \l_@@_x_final_dim \l_@@_tmpc_dim
    \dim_set_eq:NN \l_@@_y_final_dim \l_@@_tmpd_dim
  }
```

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

```
\cs_new_protected:Npn \@@_draw_standard_dotted_line:
  {
    \group_begin:
```

The dimension `\l_@@_l_dim` is the length $\ell$ of the line to draw. We use the floating point reals of the L3 programming layer to compute this length.

```
      \dim_zero_new:N \l_@@_l_dim
      \dim_set:Nn \l_@@_l_dim
        {
          \fp_to_dim:n
            {
              sqrt
               (
                 ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
                    +
                 ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
               )
            }
        }
```

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.

```
    \dim_compare:nNnT { \l_@@_l_dim } < { \c_@@_max_l_dim }
      {
        \dim_compare:nNnT { \l_@@_l_dim }  > { 1 pt }
          { \@@_draw_standard_dotted_line_i: }
      }
    \group_end:
    \bool_lazy_all:nF
      {
        { \tl_if_empty_p:N \l_@@_xdots_up_tl }
        { \tl_if_empty_p:N \l_@@_xdots_down_tl }
        { \tl_if_empty_p:N \l_@@_xdots_middle_tl }
      }
      { \@@_labels_standard_dotted_line: }
  }
\dim_const:Nn \c_@@_max_l_dim { 50 cm }
\cs_new_protected:Npn \@@_draw_standard_dotted_line_i:
  {
```

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

```
    \int_set:Nn \l_tmpa_int
      {
        \dim_ratio:nn
          {
            \l_@@_l_dim
            - \l_@@_xdots_shorten_start_dim
            - \l_@@_xdots_shorten_end_dim
          }
          { \l_@@_xdots_inter_dim }
      }
```

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

```
    \dim_set:Nn \l_tmpa_dim
```

```
      {
        ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
        \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
      }
    \dim_set:Nn \l_tmpb_dim
      {
        ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
        \dim_ratio:nn \l_@@_xdots_inter_dim \l_@@_l_dim
      }
```

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.

```
    \dim_gadd:Nn \l_@@_x_initial_dim
      {
        ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
        \dim_ratio:nn
          {
            \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
            + \l_@@_xdots_shorten_start_dim - \l_@@_xdots_shorten_end_dim
          }
          { 2 \l_@@_l_dim }
      }
    \dim_gadd:Nn \l_@@_y_initial_dim
      {
        ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
        \dim_ratio:nn
          {
            \l_@@_l_dim - \l_@@_xdots_inter_dim * \l_tmpa_int
            + \l_@@_xdots_shorten_start_dim  - \l_@@_xdots_shorten_end_dim
          }
          { 2 \l_@@_l_dim }
      }
    \pgf@relevantforpicturesizefalse
    \int_step_inline:nnn { \c_zero_int } { \l_tmpa_int }
      {
        \pgfpathcircle
          { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
          { \l_@@_xdots_radius_dim }
        \dim_add:Nn \l_@@_x_initial_dim \l_tmpa_dim
        \dim_add:Nn \l_@@_y_initial_dim \l_tmpb_dim
      }
    \pgfusepathqfill
  }
```

```
\cs_new_protected:Npn \@@_labels_standard_dotted_line:
  {
    \pgfscope
    \pgftransformshift
      {
        \pgfpointlineattime { 0.5 }
          { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
          { \pgfpoint \l_@@_x_final_dim \l_@@_y_final_dim }
      }
    \fp_set:Nn \l_tmpa_fp
      {
        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 } }
              {
                $ % $
                \scriptstyle \l_@@_xdots_middle_tl
                $ % $
              }
          }
          { }
          {
            \pgfsetfillcolor { white }
            \pgfusepath { fill }
          }
        \end { pgfscope }
      }
    \tl_if_empty:NF \l_@@_xdots_up_tl
      {
        \pgfnode
          { rectangle }
          { south }
          {
            \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
              {
                $ % $
                \scriptstyle \l_@@_xdots_up_tl
                $ % $
              }
          }
          { }
          { \pgfusepath { } }
      }
    \tl_if_empty:NF \l_@@_xdots_down_tl
      {
        \pgfnode
          { rectangle }
          { north }
          {
            \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
              {
                $ % $
                \scriptstyle \l_@@_xdots_down_tl
                $ % $
              }
          }
          { }
          { \pgfusepath { } }
      }
    \endpgfscope
  }
```

## 18 User commands available in the new environments

The commands `\@@_Ldots:`, `\@@_Cdots:`, `\@@_Vdots:`, `\@@_Ddots:` and `\@@_Iddots:` will be linked to `\Ldots`, `\Cdots`, `\Vdots`, `\Ddots` and `\Iddots` in the environments `{NiceArray}` (the other environments of nicematrix rely upon `{NiceArray}`).

The syntax of these commands uses the character `_` as embellishment and that's why we have to insert a character `_` in the *arg spec* of these commands. However, we don't know the future catcode of `_` in the main document (maybe the user will use underscore, and, in that case, the catcode is 13 because underscore activates `_`). That's why these commands will be defined in a `\hook_gput_code:nnn { begindocument } { . }` and the *arg spec* will be rescanned.

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

We rescan the *argspec* in order the correct catcode of `_` in the main document (and that's why we are in a `\AtBeginDocument`).

```
    \tl_set_rescan:Nnn \l_@@_argspec_tl { } { m  E { _ ^ : } { { } { } { } } }
    \cs_new_protected:Npn \@@_Ldots:
      { \@@_collect_options:n { \@@_Ldots_i } }
    \exp_args:NNo \NewDocumentCommand \@@_Ldots_i \l_@@_argspec_tl
      {
        \int_if_zero:nTF { \c@jCol }
          { \@@_error:nn { in~first~col } { \Ldots } }
          {
            \int_compare:nNnTF { \c@jCol } = { \l_@@_last_col_int }
              { \@@_error:nn { in~last~col } { \Ldots } }
              {
                \@@_instruction_of_type:nnn { \c_false_bool } { Ldots }
                  { #1 , down = #2 , up = #3 , middle = #4 }
              }
          }
        \bool_if:NF \l_@@_nullify_dots_bool
          { \phantom { \ensuremath { \@@_old_ldots: } } }
        \bool_gset_true:N \g_@@_empty_cell_bool
      }


    \cs_new_protected:Npn \@@_Cdots:
      { \@@_collect_options:n { \@@_Cdots_i } }
    \exp_args:NNo \NewDocumentCommand \@@_Cdots_i \l_@@_argspec_tl
      {
        \int_if_zero:nTF { \c@jCol }
          { \@@_error:nn { in~first~col } { \Cdots } }
          {
            \int_compare:nNnTF { \c@jCol } = { \l_@@_last_col_int }
              { \@@_error:nn { in~last~col } { \Cdots } }
              {
                \@@_instruction_of_type:nnn { \c_false_bool } { Cdots }
                  { #1 , down = #2 , up = #3 , middle = #4 }
              }
          }
        \bool_if:NF \l_@@_nullify_dots_bool
          { \phantom { \ensuremath { \@@_old_cdots: } } }
        \bool_gset_true:N \g_@@_empty_cell_bool
      }


    \cs_new_protected:Npn \@@_Vdots:
      { \@@_collect_options:n { \@@_Vdots_i } }
    \exp_args:NNo \NewDocumentCommand \@@_Vdots_i \l_@@_argspec_tl
      {
        \int_if_zero:nTF { \c@iRow }
          { \@@_error:nn { in~first~row } { \Vdots } }
          {
            \int_compare:nNnTF { \c@iRow } = { \l_@@_last_row_int }
              { \@@_error:nn { in~last~row } { \Vdots } }
              {
                \@@_instruction_of_type:nnn { \c_false_bool } { Vdots }
                  { #1 , down = #2 , up = #3 , middle = #4 }
              }
          }
        \bool_if:NF \l_@@_nullify_dots_bool
          { \phantom { \ensuremath { \@@_old_vdots: } } }
        \bool_gset_true:N \g_@@_empty_cell_bool
      }
```

```
  \cs_new_protected:Npn \@@_Ddots:
    { \@@_collect_options:n { \@@_Ddots_i } }
  \exp_args:NNo \NewDocumentCommand \@@_Ddots_i \l_@@_argspec_tl
    {
      \int_case:nnF \c@iRow
        {
          0                  { \@@_error:nn { in~first~row } { \Ddots } }
          \l_@@_last_row_int { \@@_error:nn { in~last~row } { \Ddots } }
        }
        {
          \int_case:nnF \c@jCol
            {
              0                  { \@@_error:nn { in~first~col } { \Ddots } }
              \l_@@_last_col_int { \@@_error:nn { in~last~col } { \Ddots } }
            }
            {
              \keys_set_known:nn { nicematrix / Ddots } { #1 }
              \@@_instruction_of_type:nnn \l_@@_draw_first_bool { Ddots }
                { #1 , down = #2 , up = #3 , middle = #4 }
            }

        }
      \bool_if:NF \l_@@_nullify_dots_bool
        { \phantom { \ensuremath { \@@_old_ddots: } } }
      \bool_gset_true:N \g_@@_empty_cell_bool
    }
```

```
  \cs_new_protected:Npn \@@_Iddots:
    { \@@_collect_options:n { \@@_Iddots_i } }
  \exp_args:NNo \NewDocumentCommand \@@_Iddots_i \l_@@_argspec_tl
    {
      \int_case:nnF \c@iRow
        {
          0                  { \@@_error:nn { in~first~row } { \Iddots } }
          \l_@@_last_row_int { \@@_error:nn { in~last~row } { \Iddots } }
        }
        {
          \int_case:nnF \c@jCol
            {
              0                  { \@@_error:nn { in~first~col } { \Iddots } }
              \l_@@_last_col_int { \@@_error:nn { in~last~col } { \Iddots } }
            }
            {
              \keys_set_known:nn { nicematrix / Ddots } { #1 }
              \@@_instruction_of_type:nnn { \l_@@_draw_first_bool } { Iddots }
                { #1 , down = #2 , up = #3 , middle = #4 }
            }
        }
      \bool_if:NF \l_@@_nullify_dots_bool
        { \phantom { \ensuremath { \@@_old_iddots: } } }
      \bool_gset_true:N \g_@@_empty_cell_bool
    }
}
```

End of the `\AddToHook`.

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

```
\keys_define:nn { nicematrix / Ddots }
  {
    draw-first .bool_set:N = \l_@@_draw_first_bool ,
    draw-first .default:n = true ,
    draw-first .value_forbidden:n = true
  }
```

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

```
\cs_new_protected:Npn \@@_Hspace:
  {
   \bool_gset_true:N \g_@@_empty_cell_bool
   \hspace
  }
```

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`.

```
\cs_new_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`.

```
\cs_new:Npn \@@_Hdotsfor:
  {
    \bool_lazy_and:nnTF
      { \int_if_zero_p:n { \c@jCol } }
      { \int_if_zero_p:n { \l_@@_first_col_int } }
      {
        \bool_if:NTF \g_@@_after_col_zero_bool
          {
            \multicolumn { 1 } { c } { }
            \@@_Hdotsfor_i:
          }
          { \@@_fatal:n { Hdotsfor~in~col~0 } }
      }
      {
        \multicolumn { 1 } { c } { }
        \@@_Hdotsfor_i:
      }
  }
```

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:`).

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

We don't put `!` before the last optional argument for homogeneity with `\Cdots`, etc. which have only one optional argument.

```
    \cs_new_protected:Npn \@@_Hdotsfor_i:
      { \@@_collect_options:n { \@@_Hdotsfor_ii } }
```

We rescan the *argspec* in order the correct catcode of _ in the main document (and that's why we are in a \AtBeginDocument).

```
    \tl_set_rescan:Nnn \l_tmpa_tl { } { m m O { } E { _ ^ : } { { } { } { } } }
    \exp_args:NNo \NewDocumentCommand \@@_Hdotsfor_ii \l_tmpa_tl
      {
        \tl_gput_right:Ne \g_@@_HVdotsfor_lines_tl
          {
            \@@_Hdotsfor:nnnn
              { \int_use:N \c@iRow }
              { \int_use:N \c@jCol }
              { #2 }
              {
                #1 , #3 ,
                down = \exp_not:n { #4 } ,
                up = \exp_not:n { #5 } ,
                middle = \exp_not:n { #6 }
              }
          }
        \prg_replicate:nn { #2 - 1 }
          {
            &
            \multicolumn { 1 } { c } { }
            \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
          }
      }
  }

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

For the row, it's easy.

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

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

```
    \int_compare:nNnTF { #2 } = { \c_one_int }
      {
        \int_set_eq:NN \l_@@_initial_j_int \c_one_int
        \bool_set_true:N \l_@@_initial_open_bool
      }
      {
        \cs_if_exist:cTF
          {
            pgf @ sh @ ns @ \@@_env:
            - \int_use:N \l_@@_initial_i_int
            - \int_eval:n { #2 - 1 }
          }
          { \int_set:Nn \l_@@_initial_j_int { #2 - 1 } }
          {
            \int_set:Nn \l_@@_initial_j_int { #2 }
            \bool_set_true:N \l_@@_initial_open_bool
          }
      }
    \int_compare:nNnTF { #2 + #3 -1 } = { \c@jCol }
      {
        \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
        \bool_set_true:N \l_@@_final_open_bool
      }
      {
        \cs_if_exist:cTF
          {
            pgf @ sh @ ns @ \@@_env:
```

```
          - \int_use:N \l_@@_final_i_int
          - \int_eval:n { #2 + #3 }
        }
        { \int_set:Nn \l_@@_final_j_int { #2 + #3 } }
        {
          \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
          \bool_set_true:N \l_@@_final_open_bool
        }
    }
  \bool_if:NT \g_@@_aux_found_bool
    {
      \group_begin:
      \@@_open_shorten:
      \int_if_zero:nTF { #1 }
        { \color { nicematrix-first-row } }
        {
          \int_compare:nNnT { #1 } = { \g_@@_row_total_int }
            { \color { nicematrix-last-row } }
        }
      \keys_set:nn { nicematrix / xdots } { #4 }
      \@@_color:o \l_@@_xdots_color_tl
      \@@_actually_draw_Ldots:
      \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:nnnn`). This declaration is done by defining a special control sequence (to nil).

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

```
\hook_gput_code:nnn { begindocument } { . }
{
  \cs_new_protected:Npn \@@_Vdotsfor:
    { \@@_collect_options:n { \@@_Vdotsfor_i } }
```

We rescan the *argspec* in order the correct catcode of `_` in the main document (and that's why we are in a `\AtBeginDocument`).

```
  \tl_set_rescan:Nnn \l_tmpa_tl { } { m m O { } E { _ ^ : } { { } { } { } } }
  \exp_args:NNo \NewDocumentCommand \@@_Vdotsfor_i \l_tmpa_tl
    {
      \bool_gset_true:N \g_@@_empty_cell_bool
      \tl_gput_right:Ne \g_@@_HVdotsfor_lines_tl
        {
          \@@_Vdotsfor:nnnn
            { \int_use:N \c@iRow }
            { \int_use:N \c@jCol }
            { #2 }
            {
              #1 , #3 ,
              down = \exp_not:n { #4 } ,
              up = \exp_not:n { #5 } ,
              middle = \exp_not:n { #6 }
            }
        }
    }
}
```

`#1` is the number of row;
`#2` is the number of column;

#3 is the numbers of rows which are involved;

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

For the column, it's easy.

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

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

```
    \int_compare:nNnTF { #1 } = { \c_one_int }
      {
        \int_set_eq:NN \l_@@_initial_i_int \c_one_int
        \bool_set_true:N \l_@@_initial_open_bool
      }
      {
        \cs_if_exist:cTF
          {
            pgf @ sh @ ns @ \@@_env:
            - \int_eval:n { #1 - 1 }
            - \int_use:N \l_@@_initial_j_int
          }
          { \int_set:Nn \l_@@_initial_i_int { #1 - 1 } }
          {
            \int_set:Nn \l_@@_initial_i_int { #1 }
            \bool_set_true:N \l_@@_initial_open_bool
          }
      }
    \int_compare:nNnTF { #1 + #3 - 1 } = { \c@iRow }
      {
        \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
        \bool_set_true:N \l_@@_final_open_bool
      }
      {
        \cs_if_exist:cTF
          {
            pgf @ sh @ ns @ \@@_env:
            - \int_eval:n { #1 + #3 }
            - \int_use:N \l_@@_final_j_int
          }
          { \int_set:Nn \l_@@_final_i_int { #1 + #3 } }
          {
            \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
            \bool_set_true:N \l_@@_final_open_bool
          }
      }

    \bool_if:NT \g_@@_aux_found_bool
      {
        \group_begin:
        \@@_open_shorten:
        \int_if_zero:nTF { #2 }
          { \color { nicematrix-first-col } }
          {
            \int_compare:nNnT { #2 } = { \g_@@_col_total_int }
              { \color { nicematrix-last-col } }
          }
        \keys_set:nn { nicematrix / xdots } { #4 }
        \@@_color:o \l_@@_xdots_color_tl
        \bool_if:NTF \l_@@_Vbrace_bool
          { \@@_actually_draw_Vbrace: }
          { \@@_actually_draw_Vdots: }
        \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:nnnn`). This declaration is done by defining a special control sequence (to nil).

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

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

```
\NewDocumentCommand \@@_rotate: { O { } }
  {
    \bool_gset_true:N \g_@@_rotate_bool
    \keys_set:nn { nicematrix / rotate } { #1 }
    \ignorespaces
  }
```

The command `\@@_rotate_p_col:` will be linked to `\rotate` in the the cells of the columns of type `p` *and al.*

```
\cs_new_protected:Npn \@@_rotate_p_col: { \@@_error:n { rotate~in~p~col } }
```

```
\keys_define:nn { nicematrix / rotate }
  {
    c .code:n = \bool_gset_true:N \g_@@_rotate_c_bool ,
    c .value_forbidden:n = true ,
    unknown .code:n = \@@_error:n { Unknown~key~for~rotate }
  }
```

## 19 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).[14]

```
\cs_new:Npn \@@_double_int_eval:n #1-#2 \q_stop
  {
    \tl_if_empty:nTF { #2 }
      { #1 }
      { \@@_double_int_eval_i:n #1-#2 \q_stop }
  }
\cs_new:Npn \@@_double_int_eval_i:n #1-#2- \q_stop
  { \int_eval:n { #1 } - \int_eval:n { #2 } }
```

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

With the following construction, the command `\@@_double_int_eval:n` is applied to both arguments before the application of `\@@_line_i:nn` (the construction uses the fact the `\@@_line_i:nn` is protected and that `\@@_double_int_eval:n` is fully expandable).

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

We rescan the *argspec* in order the correct catcode of `_` in the main document (and that's why we are in a `\AtBeginDocument`).

```
    \tl_set_rescan:Nnn \l_tmpa_tl { }
      { O { } m m ! O { } E { _ ^ : } { { } { } { } } }
    \exp_args:NNo \NewDocumentCommand \@@_line \l_tmpa_tl
      {
        \group_begin:
        \keys_set:nn { nicematrix / xdots } { #1 , #4 , down = #5 , up = #6 }
        \@@_color:o \l_@@_xdots_color_tl
        \use:e
          {
            \@@_line_i:nn
              { \@@_double_int_eval:n #2 - \q_stop }
              { \@@_double_int_eval:n #3 - \q_stop }
          }
        \group_end:
      }
  }
\cs_new_protected:Npn \@@_line_i:nn #1 #2
  {
    \bool_set_false:N \l_@@_initial_open_bool
    \bool_set_false:N \l_@@_final_open_bool
    \bool_lazy_or:nnTF
      { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #1 } }
      { \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #2 } }
      { \@@_error:nnn { unknown~cell~for~line~in~CodeAfter } { #1 } { #2 } }
```

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

```
      { \legacy_if:nF { measuring@ } { \@@_draw_line_ii:nn { #1 } { #2 } } }
  }
\hook_gput_code:nnn { begindocument } { . }
  {
    \cs_new_protected:Npe \@@_draw_line_ii:nn #1 #2
      {
```

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:`.

```
        \c_@@_pgfortikzpicture_tl
        \@@_draw_line_iii:nn { #1 } { #2 }
        \c_@@_endpgfortikzpicture_tl
      }
  }
```

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

```
\cs_new_protected:Npn \@@_draw_line_iii:nn #1 #2
  {
    \pgfrememberpicturepositiononpagetrue
    \pgfpointshapeborder { \@@_env: - #1 } { \@@_qpoint:n { #2 } }
    \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
    \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
    \pgfpointshapeborder { \@@_env: - #2 } { \@@_qpoint:n { #1 } }
    \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
    \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
    \@@_draw_line:
  }
```

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).

## 20 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
However, both arguments are implicit because they are taken by curryfication.

```
\cs_new:Npn \@@_if_row_less_than:nn { \int_compare:nNnT { \c@iRow } < }
\cs_new:Npn \@@_if_col_greater_than:nn { \int_compare:nNnF { \c@jCol } < }
```

\@@_put_in_row_style will be used several times in \RowStyle.

```
\cs_set_protected:Npn \@@_put_in_row_style:n #1
  {
    \tl_gput_right:Ne \g_@@_row_style_tl
      {
```

Be careful, \exp_not:N \@@_if_row_less_than:nn can't be replaced by a protected version of \@@_if_row_less_than:nn.

```
        \exp_not:N
        \@@_if_row_less_than:nn
          { \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).

```
          {
            \exp_not:N
            \@@_if_col_greater_than:nn
              { \int_eval:n { \c@jCol } }
              { \exp_not:n { #1 } \scan_stop: }
          }
      }
  }
\cs_generate_variant:Nn \@@_put_in_row_style:n { e }
```

```
\keys_define:nn { nicematrix / RowStyle }
  {
    cell-space-top-limit .dim_set:N = \l_tmpa_dim ,
    cell-space-top-limit .value_required:n = true ,
    cell-space-top-limit+ .code:n =
      \dim_set:Nn \l_tmpa_dim { \l_@@_cell_space_top_limit_dim + #1 } ,
    cell-space-top-limit+ .value_required:n = true ,
    cell-space-top-limit~+ .meta:n = { cell-space-top-limit+ = #1 } ,
    cell-space-bottom-limit .dim_set:N = \l_tmpb_dim ,
    cell-space-bottom-limit .value_required:n = true ,
    cell-space-bottom-limit+ .code:n =
      \dim_set:Nn \l_tmpb_dim { \l_@@_cell_space_bottom_limit_dim + #1 } ,
    cell-space-bottom-limit+ .value_required:n = true ,
    cell-space-bottom-limit~+ .meta:n = { cell-space-bottom-limit+ = #1 } ,
    cell-space-limits .meta:n =
      {
        cell-space-top-limit = #1 ,
        cell-space-bottom-limit = #1 ,
      } ,
    cell-space-limits+ .meta:n =
      {
        cell-space-top-limit += #1 ,
        cell-space-bottom-limit += #1 ,
      } ,
    cell-space-limits~+ .meta:n = { cell-space-limits+ = #1 } ,
    color .tl_set:N = \l_@@_color_tl ,
    color .value_required:n = true ,
    bold .bool_set:N = \l_@@_bold_row_style_bool ,
    bold .default:n = true ,
    nb-rows .code:n =
      \str_if_eq:eeTF { #1 } { * }
        { \int_set:Nn \l_@@_key_nb_rows_int { 500 } }
        { \int_set:Nn \l_@@_key_nb_rows_int { #1 } } ,
    nb-rows .value_required:n = true ,
    fill .tl_set:N = \l_@@_fill_tl ,
    fill .value_required:n = true ,
```

*In fine*, the opacity will be applied by `\pgfsetfillopacity`.

```
    opacity .tl_set:N = \l_@@_opacity_tl ,
    opacity .value_required:n = true ,
    rowcolor .tl_set:N = \l_@@_fill_tl ,
    rowcolor .value_required:n = true ,
    rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
    rounded-corners .default:n = 4 pt ,
    unknown .code:n =
      \@@_unknown_key:nn
        { nicematrix / RowStyle }
        { Unknown~key~for~RowStyle }
  }


\NewDocumentCommand \@@_RowStyle:n { O { } m }
  {
    \group_begin:
    \tl_clear:N \l_@@_fill_tl
    \tl_clear:N \l_@@_opacity_tl
    \tl_clear:N \l_@@_color_tl
    \int_set_eq:NN \l_@@_key_nb_rows_int \c_one_int
    \dim_zero:N \l_@@_rounded_corners_dim
    \dim_zero:N \l_tmpa_dim
    \dim_zero:N \l_tmpb_dim
    \keys_set:nn { nicematrix / RowStyle } { #1 }
```

If the key `fill` (or its alias `rowcolor`) has been used.

```
    \tl_if_empty:NF \l_@@_fill_tl
      {
        \@@_add_opacity_to_fill:
        \tl_gput_right:Ne \g_@@_pre_code_before_tl
          {
```

The command `\@@_exp_color_arg:No` is *fully expandable*.

```
            \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl
              { \int_use:N \c@iRow - \int_use:N \c@jCol }
              {
                \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int - 1 }
                - *
              }
              { \dim_use:N \l_@@_rounded_corners_dim }
          }
      }
```

```
    \@@_put_in_row_style:n { \exp_not:n { #2 } }
```

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

```
    \dim_compare:nNnT { \l_tmpa_dim } > { \c_zero_dim }
      {
        \@@_put_in_row_style:e
          {
            \tl_gput_right:Nn \exp_not:N \g_@@_cell_after_hook_tl
              {
```

It's not possible to change the following code by using \dim_set_eq:NN (because of expansion).

```
                \dim_set:Nn \l_@@_cell_space_top_limit_dim
                  { \dim_use:N \l_tmpa_dim }
              }
          }
      }
```

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

```
    \dim_compare:nNnT { \l_tmpb_dim } > { \c_zero_dim }
      {
        \@@_put_in_row_style:e
          {
            \tl_gput_right:Nn \exp_not:N \g_@@_cell_after_hook_tl
              {
                \dim_set:Nn \l_@@_cell_space_bottom_limit_dim
                  { \dim_use:N \l_tmpb_dim }
              }
          }
      }
```

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

```
    \tl_if_empty:NF \l_@@_color_tl
      {
        \@@_put_in_row_style:e
          {
            \mode_leave_vertical:
            \@@_color:n { \l_@@_color_tl }
          }
      }
```

\l_@@_bold_row_style_bool is the value of the key bold.

```
    \bool_if:NT \l_@@_bold_row_style_bool
      {
        \@@_put_in_row_style:n
          {
            \exp_not:n
              {
                \if_mode_math:
                  $ % $
                  \bfseries \boldmath
                  $ % $
                \else:
                  \bfseries \boldmath
                \fi:
              }
          }
      }
    \group_end:
    \g_@@_row_style_tl
    \ignorespaces
  }
```

The following commande must *not* be protected.

```
\cs_new:Npn \@@_rounded_from_row:n #1
  {
    \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl
```

In the following code, the "`- 1`" is *not* a subtraction.

```
      { \int_eval:n { #1 } - 1 }
      {
        \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int - 1 }
        - \exp_not:n { \int_use:N \c@jCol }
      }
      { \dim_use:N \l_@@_rounded_corners_dim }
  }
```

## 21 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).

```
\cs_new_protected:Npn \@@_add_to_colors_seq:nn #1 #2
  {
```

First, we look for the number of the color and, if it's found, we store it in `\l_tmpa_int`. If the color is not present in `\l_@@_colors_seq`, `\l_tmpa_int` will remain equal to 0.

```
    \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. `\str_if_in:nnF` is mandatory: don't use `\tl_if_in:nnF`.

```
    \str_if_in:nnF { #1 } { !! }
      {
        \seq_map_indexed_inline:Nn \g_@@_colors_seq
```

We use `\str_if_eq:eeTF` which is slightly faster than `\tl_if_eq:nnTF`.

```
          { \str_if_eq:eeT { #1 } { ##2 } { \int_set:Nn \l_tmpa_int { ##1 } } }
      }
    \int_if_zero:nTF { \l_tmpa_int }
```

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

```
      {
        \seq_gput_right:Nn \g_@@_colors_seq { #1 }
        \tl_gset:ce { g_@@_color _ \seq_count:N \g_@@_colors_seq _ tl } { #2 }
      }
```

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

```
    { \tl_gput_right:ce { g_@@_color _ \int_use:N \l_tmpa_int _tl } { #2 } }
  }
\cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e }
\cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e e }
```

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

```
\cs_new_protected:Npn \@@_clip_with_rounded_corners:
  {
    \dim_compare:nNnT { \l_@@_tab_rounded_corners_dim } > { \c_zero_dim }
      {
```

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

```
        \group_begin:
        \pgfsetcornersarced
          {
            \pgfpoint
              { \l_@@_tab_rounded_corners_dim }
              { \l_@@_tab_rounded_corners_dim }
          }
```

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.

```
        \bool_if:NTF \l_@@_hvlines_bool
          {
            \pgfpathrectanglecorners
              {
                \pgfpointadd
                  { \@@_qpoint:n { row-1 } }
                  { \pgfpoint { 0.5 \arrayrulewidth } { \c_zero_dim } }
              }
              {
                \pgfpointadd
                  {
                    \@@_qpoint:n
                      { \int_eval:n { \int_max:nn { \c@iRow } { \c@jCol } + 1 } }
                  }
                  { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
              }
          }
          {
            \pgfpathrectanglecorners
              { \@@_qpoint:n { row-1 } }
              {
                \pgfpointadd
                  {
                    \@@_qpoint:n
                      { \int_eval:n { \int_max:nn { \c@iRow } { \c@jCol } + 1 } }
                  }
                  { \pgfpoint \c_zero_dim \arrayrulewidth }
              }
          }
        \pgfusepath { clip }
        \group_end:
```

The TeX group was for `\pgfsetcornersarced`.

```
      }
  }
```

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`).

```
\cs_new_protected:Npn \@@_actually_color:
  {
    \pgfpicture
    \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.

```
    \@@_clip_with_rounded_corners:
    \seq_map_indexed_inline:Nn \g_@@_colors_seq
      {
        \int_compare:nNnTF { ##1 } = { \c_one_int }
          {
            \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_nocolor:n
            \use:c { g_@@_color _ 1 _tl }
            \cs_set_eq:NN \@@_cartesian_path:n \@@_cartesian_path_normal:n
          }
          {
            \begin { pgfscope }
              \@@_color_opacity: ##2
              \use:c { g_@@_color _ ##1 _tl }
              \tl_gclear:c { g_@@_color _ ##1 _tl }
              \pgfusepath { fill }
            \end { pgfscope }
          }
      }
    \endpgfpicture
  }
```

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

```
\cs_new_protected:Npn \@@_color_opacity:
  {
    \peek_meaning:NTF [
      { \@@_color_opacity:w }
      { \@@_color_opacity:w [ ] }
  }
```

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.

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

```
    \tl_if_empty:NF \l_tmpa_tl { \exp_args:No \pgfsetfillopacity \l_tmpa_tl }
    \tl_if_empty:NTF \l_tmpb_tl
      { \@declaredcolor }
      { \use:e { \exp_not:N \@undeclaredcolor [ \l_tmpb_tl ] } }
  }
```

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

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

Here, we use `\def` instead of `\tl_set:Nn` for efficiency only.

```
\cs_new_protected:Npn \@@_cartesian_color:nn #1 #2
  {
    \def \l_@@_rows_tl { #1 }
    \def \l_@@_cols_tl { #2 }
    \@@_cartesian_path:
  }
```

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

```
\NewDocumentCommand \@@_rowcolor { O { } m m }
  {
    \tl_if_blank:nF { #2 }
      {
        \@@_add_to_colors_seq:en
          { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
          { \@@_cartesian_color:nn { #3 } { - } }
      }
  }
```

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

```
\NewDocumentCommand \@@_columncolor { O { } m m }
  {
    \tl_if_blank:nF { #2 }
      {
        \@@_add_to_colors_seq:en
          { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
          { \@@_cartesian_color:nn { - } { #3 } }
      }
  }
```

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

```
\NewDocumentCommand \@@_rectanglecolor { O { } m m m }
  {
    \tl_if_blank:nF { #2 }
      {
        \@@_add_to_colors_seq:en
          { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
          { \@@_rectanglecolor:nnn { #3 } { #4 } { \c_zero_dim } }
      }
  }
```

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

```
\NewDocumentCommand \@@_roundedrectanglecolor { O { } m m m m }
  {
    \tl_if_blank:nF { #2 }
      {
        \@@_add_to_colors_seq:en
          { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
          { \@@_rectanglecolor:nnn { #3 } { #4 } { #5 } }
      }
  }
```

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

```
\cs_new_protected:Npn \@@_rectanglecolor:nnn #1 #2 #3
  {
    \@@_cut_on_hyphen:w #1 \q_stop
    \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
    \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
    \@@_cut_on_hyphen:w #2 \q_stop
    \tl_set:Ne \l_@@_rows_tl { \l_@@_tmpc_tl - \l_tmpa_tl }
    \tl_set:Ne \l_@@_cols_tl { \l_@@_tmpd_tl - \l_tmpb_tl }
```

The command `\@@_cartesian_path:n` takes in two implicit arguments: `\l_@@_cols_tl` and `\l_@@_rows_tl`.

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

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

```
\NewDocumentCommand \@@_cellcolor { O { } m m }
  {
    \clist_map_inline:nn { #3 }
      { \@@_rectanglecolor [ #1 ] { #2 } { ##1 } { ##1 } }
  }
```

```
\NewDocumentCommand \@@_chessboardcolors { O { } m m  }
  {
    \int_step_inline:nn { \c@iRow }
      {
        \int_step_inline:nn { \c@jCol }
          {
            \int_if_even:nTF { ####1 + ##1 }
              { \@@_cellcolor [ #1 ] { #2 } }
              { \@@_cellcolor [ #1 ] { #3 } }
            { ##1 - ####1 }
          }
      }
  }
```

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".

```
\NewDocumentCommand \@@_arraycolor { O { } m }
  {
    \@@_rectanglecolor [ #1 ] { #2 }
      { 1 - 1 }
      { \int_use:N \c@iRow - \int_use:N \c@jCol }
  }
```

```
\keys_define:nn { nicematrix / rowcolors }
  {
    respect-blocks .bool_set:N = \l_@@_respect_blocks_bool ,
    respect-blocks .default:n = true ,
    cols .tl_set:N = \l_@@_cols_tl ,
    restart .bool_set:N = \l_@@_rowcolors_restart_bool ,
    restart .default:n = true ,
    unknown .code:n = \@@_error:n { Unknown~key~for~rowcolors }
  }
```

The command `\rowcolors` (accessible in the `\CodeBefore`) is inspired by the command `\rowcolors` of the package xcolor (with the option `table`). However, the command `\rowcolors` of nicematrix has *not* the optional argument of the command `\rowcolors` of xcolor.
Here is an example: `\rowcolors{1}{blue!10}{}[respect-blocks]`.
In nicematrix, the command `\@@_rowcolors` 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*.

```
\NewDocumentCommand \@@_rowlistcolors { O { } m m O { } }
  {
```

`\l_@@_colors_seq` will be the list of colors.

```
    \pgfpicture
    \pgf@relevantforpicturesizefalse
    \seq_clear_new:N \l_@@_colors_seq
    \seq_set_split:Nnn \l_@@_colors_seq { , } { #3 }
    \tl_clear_new:N \l_@@_cols_tl
    \tl_set:Nn \l_@@_cols_tl { - }
    \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).

```
    \int_zero_new:N \l_@@_color_int
    \int_set_eq:NN \l_@@_color_int \c_one_int
    \bool_if:NT \l_@@_respect_blocks_bool
      {
```

We don't want to take into account a block which is entirely in the "first column" (number 0) or in the "last column" and that's why we filter the sequence of the blocks (in the sequence `\l_tmpa_seq`).

```
        \seq_set_eq:NN \l_tmpb_seq \g_@@_pos_of_blocks_seq
        \seq_set_filter:NNn \l_tmpa_seq \l_tmpb_seq
          { \@@_not_in_exterior_p:nnnnn ##1 }
      }
```

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

```
    \clist_map_inline:nn { #2 }
      {
        \tl_set:Nn \l_tmpa_tl { ##1 }
        \tl_if_in:NnTF \l_tmpa_tl { - }
          { \@@_cut_on_hyphen:w ##1 \q_stop }
          { \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.

```
        \int_set:Nn \l_tmpa_int \l_tmpa_tl
        \int_set:Nn \l_@@_color_int
          { \bool_if:NTF \l_@@_rowcolors_restart_bool { 1 } { \l_tmpa_tl } }
        \int_set:Nn \l_@@_tmpc_int \l_tmpb_tl
        \int_do_until:nNnn \l_tmpa_int > \l_@@_tmpc_int
          {
```

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

```
            \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).

```
            \bool_if:NT \l_@@_respect_blocks_bool
              {
                \seq_set_filter:NNn \l_tmpb_seq \l_tmpa_seq
                  { \@@_intersect_our_row_p:nnnnn ####1 }
                \seq_map_inline:Nn \l_tmpb_seq { \@@_rowcolors_i:nnnnn ####1 }
```

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

```
              }
            \tl_set:Ne \l_@@_rows_tl
              { \int_use:N \l_tmpa_int - \int_use:N \l_tmpb_int }
```

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

```
            \tl_set:Ne \l_@@_color_tl
              {
                \@@_color_index:n
                  {
                    \int_mod:nn
                      { \l_@@_color_int - 1 }
                      { \seq_count:N \l_@@_colors_seq }
                    + 1
                  }
              }
            \tl_if_empty:NF \l_@@_color_tl
              {
                \@@_add_to_colors_seq:ee
                  { \tl_if_blank:nF { #1 } { [ #1 ] } { \l_@@_color_tl } }
                  { \@@_cartesian_color:nn { \l_@@_rows_tl } { \l_@@_cols_tl } }
              }
            \int_incr:N \l_@@_color_int
            \int_set:Nn \l_tmpa_int { \l_tmpb_int + 1 }
          }
      }
    \endpgfpicture
  }
```

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.

```
\cs_new:Npn \@@_color_index:n #1
  {
```

Be careful: this command `\@@_color_index:n` must be "*fully expandable*".

```
    \str_if_eq:eeTF { \seq_item:Nn \l_@@_colors_seq { #1 } } { = }
      { \@@_color_index:n { #1 - 1 } }
      { \seq_item:Nn \l_@@_colors_seq { #1 } }
  }
```

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.

```
\NewDocumentCommand \@@_rowcolors { O { } m m m }
  { \@@_rowlistcolors [ #1 ] { #2 } { { #3 } , { #4 } } }
```

The braces around `#3` and `#4` are mandatory.

```
\cs_new_protected:Npn \@@_rowcolors_i:nnnnn #1 #2 #3 #4 #5
  {
    \int_compare:nNnT { #3 } > { \l_tmpb_int }
      { \int_set:Nn \l_tmpb_int { #3 } }
  }
```

```
\prg_new_conditional:Nnn \@@_not_in_exterior:nnnnn { p }
  {
    \int_if_zero:nTF { #4 }
      { \prg_return_false: }
      {
        \int_compare:nNnTF { #2 } > { \c@jCol }
          { \prg_return_false: }
          { \prg_return_true: }
      }
  }
```

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

```
\prg_new_conditional:Nnn \@@_intersect_our_row:nnnnn { p }
  {
    \int_compare:nNnTF { #1 } > { \l_tmpa_int }
      { \prg_return_false: }
      {
        \int_compare:nNnTF { \l_tmpa_int } > { #3 }
          { \prg_return_false: }
          { \prg_return_true: }
      }
  }
```

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`).

```
\cs_new_protected:Npn \@@_cartesian_path_normal:n #1
  {
    \dim_compare:nNnTF { #1 } = { \c_zero_dim }
      {
        \bool_if:NTF \l_@@_nocolor_used_bool
          { \@@_cartesian_path_normal_ii: }
          {
            \clist_if_empty:NTF \l_@@_corners_cells_clist
              { \@@_cartesian_path_normal_i:n { #1 } }
              { \@@_cartesian_path_normal_ii: }
          }
      }
      { \@@_cartesian_path_normal_i:n { #1 } }
  }
```

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.

```
\cs_new_protected:Npn \@@_cartesian_path_normal_i:n #1
  {
    \pgfsetcornersarced { \pgfpoint { #1 } { #1 } }
```

We begin the loop over the columns.

```
    \clist_map_inline:Nn \l_@@_cols_tl
      {
```

We use `\def` instead of `\tl_set:Nn` for efficiency only.

```
        \def \l_tmpa_tl { ##1 }
        \tl_if_in:NnTF \l_tmpa_tl { - }
          { \@@_cut_on_hyphen:w ##1 \q_stop }
          { \def \l_tmpb_tl { ##1 } }
        \tl_if_empty:NTF \l_tmpa_tl
          { \def \l_tmpa_tl { 1 } }
          {
            \str_if_eq:eeT { \l_tmpa_tl } { * }
              { \def \l_tmpa_tl { 1 } }
          }
        \int_compare:nNnT { \l_tmpa_tl } > { \g_@@_col_total_int }
          { \@@_error:n { Invalid~col~number } }
        \tl_if_empty:NTF \l_tmpb_tl
          { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
          {
            \str_if_eq:eeT { \l_tmpb_tl } { * }
              { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
          }
        \int_compare:nNnT { \l_tmpb_tl } > { \g_@@_col_total_int }
          { \tl_set:No \l_tmpb_tl { \int_use:N \g_@@_col_total_int } }
```

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

```
        \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
        \@@_qpoint:n { col - \l_tmpa_tl }
        \int_compare:nNnTF { \l_@@_first_col_int } = { \l_tmpa_tl }
          { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
          { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
        \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 }  }
        \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }
```

We begin the loop over the rows. We use \def instead of \tl_set:Nn for efficiency only.

```
        \clist_map_inline:Nn \l_@@_rows_tl
          {
            \def \l_tmpa_tl { ####1 }
            \tl_if_in:NnTF \l_tmpa_tl { - }
              { \@@_cut_on_hyphen:w ####1 \q_stop }
              { \@@_cut_on_hyphen:w ####1 - ####1 \q_stop }
            \tl_if_empty:NTF \l_tmpa_tl
              { \def \l_tmpa_tl { 1 } }
              {
                \str_if_eq:eeT { \l_tmpa_tl } { * }
                  { \def \l_tmpa_tl { 1 } }
              }
            \tl_if_empty:NTF \l_tmpb_tl
              { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }
              {
                \str_if_eq:eeT { \l_tmpb_tl } { * }
                  { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }
              }
            \int_compare:nNnT { \l_tmpa_tl } > { \g_@@_row_total_int }
              { \@@_error:n { Invalid~row~number } }
            \int_compare:nNnT { \l_tmpb_tl } > { \g_@@_row_total_int }
              { \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.

```
            \cs_if_exist:cF
              { @@ _ nocolor _ \l_tmpa_tl - \l_@@_tmpc_tl }
              {
                \@@_qpoint:n { row - \int_eval:n { \l_tmpb_tl + 1 } }
                \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
                \@@_qpoint:n { row - \l_tmpa_tl }
                \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
                \pgfpathrectanglecorners
                  { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
                  { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
              }
          }
      }
  }
```

Now, the case where the cells will be colored cell by cell (it's mandatory for example if the key corners is used).

```
\cs_new_protected:Npn \@@_cartesian_path_normal_ii:
  {
    \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
    \@@_expand_clist:NN \l_@@_rows_tl \c@iRow
```

We begin the loop over the columns.

```
    \clist_map_inline:Nn \l_@@_cols_tl
      {
        \@@_qpoint:n { col - ##1 }
        \int_compare:nNnTF { \l_@@_first_col_int } = { ##1 }
          { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
          { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
        \@@_qpoint:n { col - \int_eval:n { ##1 + 1 }  }
        \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }
```

We begin the loop over the rows.

```
        \clist_map_inline:Nn \l_@@_rows_tl
          {
            \@@_if_in_corner:nF { ####1 - ##1 }
              {
                \@@_qpoint:n { row - \int_eval:n { ####1 + 1 } }
                \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
                \@@_qpoint:n { row - ####1 }
                \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
                \cs_if_exist:cF { @@ _ nocolor _ ####1 - ##1 }
                  {
                    \pgfpathrectanglecorners
                      { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
                      { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
                  }
              }
          }
      }
  }
```

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`).

```
\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

```
\cs_new_protected:Npn \@@_cartesian_path_nocolor:n #1
  {
    \bool_set_true:N \l_@@_nocolor_used_bool
    \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
    \@@_expand_clist:NN \l_@@_rows_tl \c@iRow
```

We begin the loop over the columns.

```
    \clist_map_inline:Nn \l_@@_rows_tl
      {
        \clist_map_inline:Nn \l_@@_cols_tl
          { \cs_set_nopar:cpn { @@ _ nocolor _ ##1 - ####1 } { } }
      }
  }
```

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`.

```
\cs_new_protected:Npn \@@_expand_clist:NN #1 #2
  {
    \clist_set_eq:NN \l_tmpa_clist #1
    \clist_clear:N #1
    \clist_map_inline:Nn \l_tmpa_clist
      {
```

We use `\def` instead of `\tl_set:Nn` for efficiency only.

```
        \def \l_tmpa_tl { ##1 }
        \tl_if_in:NnTF \l_tmpa_tl { - }
          { \@@_cut_on_hyphen:w ##1 \q_stop }
          { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
        \bool_lazy_or:nnT
          { \str_if_eq_p:ee { \l_tmpa_tl } { * } }
          { \tl_if_blank_p:o \l_tmpa_tl }
          { \def \l_tmpa_tl { 1 } }
        \bool_lazy_or:nnT
          { \str_if_eq_p:ee { \l_tmpb_tl } { * } }
          { \tl_if_blank_p:o \l_tmpb_tl }
          { \tl_set:No \l_tmpb_tl { \int_use:N #2 } }
        \int_compare:nNnT { \l_tmpb_tl } > { #2 }
          { \tl_set:No \l_tmpb_tl { \int_use:N #2 } }
        \int_step_inline:nnn { \l_tmpa_tl } { \l_tmpb_tl }
          { \clist_put_right:Nn #1 { ####1 } }
      }
  }
```

The following command will be linked to `\cellcolor` in the tabular.

```
\NewDocumentCommand \@@_cellcolor_tabular { O { } m }
  {
    \tl_gput_right:Ne \g_@@_pre_code_before_tl
      {
```

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).

```
        \@@_cellcolor [ #1 ] { \exp_not:n { #2 } }
          { \int_use:N \c@iRow - \int_use:N \c@jCol }
      }
    \ignorespaces
  }
\NewDocumentCommand \@@_cellcolor_error { O { } m }
  { \@@_error:n { cellcolor~in~Block } }
% \end{macrocode}
%
%    \begin{macrocode}
\NewDocumentCommand \@@_rowcolor_error { O { } m }
  { \@@_error:n { rowcolor~in~Block } }
% \end{macrocode}
%
% \bigskip
% The following command will be linked to |\rowcolor| in the tabular.
%    \begin{macrocode}
\NewDocumentCommand \@@_rowcolor_tabular { O { } m }
  {
    \tl_gput_right:Ne \g_@@_pre_code_before_tl
      {
        \@@_rectanglecolor [ #1 ] { \exp_not:n { #2 } }
          { \int_use:N \c@iRow - \int_use:N \c@jCol }
          { \int_use:N \c@iRow - \exp_not:n { \int_use:N \c@jCol } }
      }
    \ignorespaces
  }
```

The following command will be linked to `\rowcolors` in the tabular. The last argument (an optional argument between square brackets is taken by curryfication).

```
\NewDocumentCommand { \@@_rowcolors_tabular } { O { } m m }
  { \@@_rowlistcolors_tabular [ #1 ] { { #2 } , { #3 } } }
```

The braces around `#2` and `#3` are mandatory.

The following command will be linked to `\rowlistcolors` in the tabular.

```
\NewDocumentCommand { \@@_rowlistcolors_tabular } { O { } m O { } }
  {
```

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`.

```
    \seq_gclear:N \g_tmpa_seq
    \seq_map_inline:Nn \g_@@_rowlistcolors_seq
      { \@@_rowlistcolors_tabular:nnnn ##1 }
    \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`.

```
    \seq_gput_right:Ne \g_@@_rowlistcolors_seq
      {
        { \int_use:N \c@iRow }
        { \exp_not:n { #1 } }
        { \exp_not:n { #2 } }
        { restart , cols = \int_use:N \c@jCol - , \exp_not:n { #3 } }
      }
    \ignorespaces
  }
```

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`).

```
\cs_new_protected:Npn \@@_rowlistcolors_tabular:nnnn #1 #2 #3 #4
  {
    \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).

```
      { \seq_gput_right:Nn \g_tmpa_seq { { #1 } { #2 } { #3 } { #4 } } }
      {
        \tl_gput_right:Ne \g_@@_pre_code_before_tl
          {
            \@@_rowlistcolors
               [ \exp_not:n { #2 } ]
               { #1 - \int_eval:n { \c@iRow - 1 } }
               { \exp_not:n { #3 } }
               [ \exp_not:n { #4 } ]
          }
      }
  }
```

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.

```
\cs_new_protected:Npn \@@_clear_rowlistcolors_seq:
  {
    \seq_map_inline:Nn \g_@@_rowlistcolors_seq
      { \@@_rowlistcolors_tabular_ii:nnnn ##1 }
    \seq_gclear:N \g_@@_rowlistcolors_seq
  }

\cs_new_protected:Npn \@@_rowlistcolors_tabular_ii:nnnn #1 #2 #3 #4
  {
    \tl_gput_right:Nn \g_@@_pre_code_before_tl
      { \@@_rowlistcolors [ #2 ] { #1 } { #3 } [ #4 ] }
  }
```

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.

```
\NewDocumentCommand \@@_columncolor_preamble { O { } m }
  {
```

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).

```
    \int_compare:nNnT { \c@jCol } > { \g_@@_col_total_int }
      {
```

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).

```
          \tl_gput_left:Ne \g_@@_pre_code_before_tl
            {
              \exp_not:N \columncolor [ #1 ]
                { \exp_not:n { #2 } } { \int_use:N \c@jCol }
            }
        }
    }
```

```
\cs_new_protected:Npn \@@_EmptyColumn:n #1
  {
    \clist_map_inline:nn { #1 }
      {
        \seq_gput_right:Nn \g_@@_future_pos_of_blocks_seq
          { { -2 } { #1 } { 98 } { ##1 } { } } % 98 and not 99 !
        \columncolor { nocolor } { ##1 }
      }
  }
```

```
\cs_new_protected:Npn \@@_EmptyRow:n #1
  {
    \clist_map_inline:nn { #1 }
      {
        \seq_gput_right:Nn \g_@@_future_pos_of_blocks_seq
          { { ##1 } { -2 } { ##1 } { 98 } { } } % 98 and not 99 !
        \rowcolor { nocolor } { ##1 }
      }
  }
```

## 22 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`.

```
\cs_new_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`.

```
\cs_new_protected:Npn \@@_OnlyMainNiceMatrix:n #1
  {
    \int_if_zero:nTF { \l_@@_first_col_int }
      { \@@_OnlyMainNiceMatrix_i:n { #1 } }
      {
        \int_if_zero:nTF { \c@jCol }
          {
            \int_compare:nNnF { \c@iRow } = { -1 }
              {
                \int_compare:nNnF { \c@iRow } = { \l_@@_last_row_int - 1 }
                  { #1 }
              }
          }
          { \@@_OnlyMainNiceMatrix_i:n { #1 } }
      }
  }
```

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.

```
\cs_new_protected:Npn \@@_OnlyMainNiceMatrix_i:n #1
  {
    \int_if_zero:nF { \c@iRow }
      {
        \int_compare:nNnF { \c@iRow } = { \l_@@_last_row_int }
          {
            \int_compare:nNnT { \c@jCol } > { \c_zero_int }
              { \bool_if:NF \l_@@_in_last_col_bool { #1 } }
          }
      }
  }
```

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`).

The following command will be used for `\Toprule`, `\BottomRule` and `\MidRule`.

```
\cs_new:Npn \@@_tikz_booktabs_loaded:nn #1 #2
  {
    \IfPackageLoadedTF { tikz }
      {
        \IfPackageLoadedTF { booktabs }
          { #2 }
          { \@@_error:nn { TopRule~without~booktabs } { #1 } }
      }
      { \@@_error:nn { TopRule~without~tikz } { #1 } }
  }
\NewExpandableDocumentCommand { \@@_TopRule } { }
  { \@@_tikz_booktabs_loaded:nn { \TopRule } { \@@_TopRule_i: } }
\cs_new:Npn \@@_TopRule_i:
  {
    \noalign \bgroup
      \peek_meaning:NTF [
        { \@@_TopRule_ii: }
        { \@@_TopRule_ii: [ \dim_use:N \heavyrulewidth ] }
  }
\NewDocumentCommand \@@_TopRule_ii: { o }
  {
    \tl_gput_right:Ne \g_@@_pre_code_after_tl
      {
        \@@_hline:n
          {
            position = \int_eval:n { \c@iRow + 1 } ,
            tikz =
              {
                line~width = #1 ,
                yshift =  0.25 \arrayrulewidth ,
                shorten~< = - 0.5 \arrayrulewidth
              } ,
            total-width = #1
          }
      }
    \skip_vertical:n { \belowrulesep + #1 }
    \egroup
  }
\NewExpandableDocumentCommand { \@@_BottomRule } { }
  { \@@_tikz_booktabs_loaded:nn { \BottomRule } { \@@_BottomRule_i: } }
\cs_new:Npn \@@_BottomRule_i:
  {
    \noalign \bgroup
      \peek_meaning:NTF [
        { \@@_BottomRule_ii: }
        { \@@_BottomRule_ii: [ \dim_use:N \heavyrulewidth ] }
  }
\NewDocumentCommand \@@_BottomRule_ii: { o }
  {
    \tl_gput_right:Ne \g_@@_pre_code_after_tl
      {
        \@@_hline:n
          {
            position = \int_eval:n { \c@iRow + 1 } ,
            tikz =
              {
                line~width = #1 ,
                yshift =  0.25 \arrayrulewidth ,
                shorten~< = - 0.5 \arrayrulewidth
              } ,
            total-width = #1 ,
          }
      }
    \skip_vertical:N \aboverulesep
    \@@_create_row_node_i:
    \skip_vertical:n { #1 }
    \egroup
  }
\NewExpandableDocumentCommand { \@@_MidRule } { }
  { \@@_tikz_booktabs_loaded:nn { \MidRule } { \@@_MidRule_i: } }
\cs_new:Npn \@@_MidRule_i:
  {
    \noalign \bgroup
      \peek_meaning:NTF [
        { \@@_MidRule_ii: }
        { \@@_MidRule_ii: [ \dim_use:N \lightrulewidth ] }
  }
\NewDocumentCommand \@@_MidRule_ii: { o }
  {
    \skip_vertical:N \aboverulesep
    \@@_create_row_node_i:
    \tl_gput_right:Ne \g_@@_pre_code_after_tl
      {
        \@@_hline:n
          {
            position = \int_eval:n { \c@iRow + 1 } ,
            tikz =
              {
                line~width = #1 ,
                yshift =  0.25 \arrayrulewidth ,
                shorten~< = - 0.5 \arrayrulewidth
              } ,
            total-width = #1 ,
          }
      }
    \skip_vertical:n { \belowrulesep + #1 }
    \egroup
  }
```

### 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.

```
\keys_define:nn { nicematrix / Rules }
  {
    position .int_set:N = \l_@@_position_int ,
    position .value_required:n = true ,
    start .int_set:N = \l_@@_start_int ,
    end .code:n =
      \bool_lazy_or:nnTF
        { \tl_if_empty_p:n { #1 } }
        { \str_if_eq_p:ee { #1 } { last } }
        { \int_set_eq:NN \l_@@_end_int \c@jCol }
        { \int_set:Nn \l_@@_end_int { #1 } }
  }
```

It's possible that the rule won't be drawn continuously from `start` to `end` because of the blocks (created with the command `\Block`), the virtual blocks (created by `\Cdots`, etc.), etc. That's why an analysis 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.

```
\keys_define:nn { nicematrix / RulesBis }
  {
    multiplicity .int_set:N = \l_@@_multiplicity_int ,
    multiplicity .initial:n = 1 ,
    dotted .bool_set:N = \l_@@_dotted_bool ,
    dotted .initial:n = false ,
    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]`.

```
    color .code:n =
      \@@_set_CTarc:n { #1 }
      \tl_set:Nn \l_@@_rule_color_tl { #1 } ,
    color .value_required:n = true ,
    sep-color .code:n = \@@_set_CTdrsc:n { #1 } ,
    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.

```
    tikz .code:n =
      \IfPackageLoadedTF { tikz }
        { \clist_put_right:Nn \l_@@_tikz_rule_tl { #1 } }
        { \@@_error:n { tikz~without~tikz } } ,
    tikz .value_required:n = true ,
    total-width .dim_set:N = \l_@@_rule_width_dim ,
    total-width .value_required:n = true ,
    width .meta:n = { total-width = #1 } ,
    unknown .code:n =
      \@@_unknown_key:nn
```

```
        { nicematrix / RulesBis }
        { Unknown~key~for~RulesBis }
  }
```

## The vertical rules

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

```
\cs_new_protected:Npn \@@_vline:n #1
  {
```

The group is for the options.

```
    \group_begin:
    \int_set_eq:NN \l_@@_end_int \c@iRow
    \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).

```
    \int_compare:nNnT { \l_@@_position_int } < { \c@jCol + 2 }
      \@@_vline_i:
    \group_end:
  }
\cs_new_protected:Npn \@@_vline_i:
  {
```

`\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`.

```
    \tl_set:No \l_tmpb_tl { \int_use:N \l_@@_position_int }
    \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
      \l_tmpa_tl
      {
```

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.

```
        \bool_gset_true:N \g_tmpa_bool
        \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
          { \@@_test_vline_in_block:nnnnn ##1 }
        \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
          { \@@_test_vline_in_block:nnnnn ##1 }
        \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
          { \@@_test_vline_in_stroken_block:nnnn ##1 }
        \clist_if_empty:NF \l_@@_corners_clist { \@@_test_in_corner_v: }
        \bool_if:NTF \g_tmpa_bool
          {
            \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.

```
              { \int_set:Nn \l_@@_local_start_int \l_tmpa_tl }
          }
          {
            \int_compare:nNnT { \l_@@_local_start_int } > { \c_zero_int }
              {
                \int_set:Nn \l_@@_local_end_int { \l_tmpa_tl - 1 }
                \@@_vline_ii:
                \int_zero:N \l_@@_local_start_int
              }
          }
      }
    \int_compare:nNnT { \l_@@_local_start_int } > { \c_zero_int }
      {
        \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
        \@@_vline_ii:
      }
  }
```

```
\cs_new_protected:Npn \@@_test_in_corner_v:
   {
     \int_compare:nNnTF { \l_tmpb_tl } = { \c@jCol + 1 }
       {
         \@@_if_in_corner:nT { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
           { \bool_set_false:N \g_tmpa_bool }
       }
       {
         \@@_if_in_corner:nT { \l_tmpa_tl - \l_tmpb_tl }
           {
             \int_compare:nNnTF { \l_tmpb_tl } = { \c_one_int }
               { \bool_set_false:N \g_tmpa_bool }
               {
                 \@@_if_in_corner:nT
                   { \l_tmpa_tl - \int_eval:n { \l_tmpb_tl - 1 } }
                   { \bool_set_false:N \g_tmpa_bool }
               }
           }
       }
   }
```

```
\cs_new_protected:Npn \@@_vline_ii:
  {
    \tl_clear:N \l_@@_tikz_rule_tl
    \keys_set:no { nicematrix / RulesBis } \l_@@_other_keys_tl
    \bool_if:NTF \l_@@_dotted_bool
      { \@@_vline_iv: }
      {
        \tl_if_empty:NTF \l_@@_tikz_rule_tl
          { \@@_vline_iii: }
          { \@@_vline_v: }
      }
  }
```

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

```
\cs_new_protected:Npn \@@_vline_iii:
  {
    \pgfpicture
    \pgfrememberpicturepositiononpagetrue
    \pgf@relevantforpicturesizefalse
    \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
    \dim_set_eq:NN \l_tmpa_dim \pgf@y
    \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
    \dim_set:Nn \l_tmpb_dim
      {
        \pgf@x
        - 0.5 \l_@@_rule_width_dim
        +
        ( \arrayrulewidth * \l_@@_multiplicity_int
           + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
      }
    \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
    \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
    \bool_lazy_all:nT
      {
        { \int_compare_p:nNn { \l_@@_multiplicity_int } > { \c_one_int } }
        { \cs_if_exist_p:N \CT@drsc@ }
        { ! \tl_if_blank_p:o \CT@drsc@ }
      }
      {
        \group_begin:
        \CT@drsc@
        \dim_add:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }
        \dim_sub:Nn \l_@@_tmpc_dim { 0.5 \arrayrulewidth }
        \dim_set:Nn \l_@@_tmpd_dim
          {
            \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
            * ( \l_@@_multiplicity_int - 1 )
          }
        \pgfpathrectanglecorners
          { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
          { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
        \pgfusepath { fill }
        \group_end:
      }
    \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
    \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
    \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
      {
        \dim_sub:Nn \l_tmpb_dim { \arrayrulewidth + \doublerulesep }
        \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
        \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
      }
    \CT@arc@
    \pgfsetlinewidth { 1.1 \arrayrulewidth }
    \pgfsetrectcap
    \pgfusepathqstroke
    \endpgfpicture
  }
```

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

```
\cs_new_protected:Npn \@@_vline_iv:
  {
    \pgfpicture
    \pgfrememberpicturepositiononpagetrue
    \pgf@relevantforpicturesizefalse
    \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
    \dim_set:Nn \l_@@_x_initial_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
    \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
    \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
    \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
    \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
    \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
    \CT@arc@
    \@@_draw_line:
    \endpgfpicture
  }
```

The following code is for the case when the user uses the key `tikz`.

```
\cs_new_protected:Npn \@@_vline_v:
  {
    \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.

```
    \CT@arc@
    \tl_if_empty:NF \l_@@_rule_color_tl
      { \tl_put_right:Ne \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
    \pgfrememberpicturepositiononpagetrue
    \pgf@relevantforpicturesizefalse
    \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
    \dim_set_eq:NN \l_tmpa_dim \pgf@y
    \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
    \dim_set:Nn \l_tmpb_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
    \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
    \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
    \exp_args:No \tikzset \l_@@_tikz_rule_tl
    \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
      ( \l_tmpb_dim , \l_tmpa_dim ) --
      ( \l_tmpb_dim , \l_@@_tmpc_dim ) ;
    \end { tikzpicture }
  }
```

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).

```
\cs_new_protected:Npn \@@_draw_vlines:
  {
    \int_step_inline:nnn
      {
        \bool_lazy_or:nnTF { \g_@@_delims_bool } { \l_@@_except_borders_bool }
          { 2 }
          { 1 }
       }
      {
        \bool_lazy_or:nnTF { \g_@@_delims_bool } { \l_@@_except_borders_bool }
          { \c@jCol }
          { \int_eval:n { \c@jCol + 1 } }
      }
      {
        \str_if_eq:eeF { \l_@@_vlines_clist } { all }
          { \clist_if_in:NnT \l_@@_vlines_clist { ##1 } }
          { \@@_vline:n { position = ##1 , total-width = \arrayrulewidth } }
      }
  }
```

### 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}`.

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

The group is for the options.

```
    \group_begin:
    \int_set_eq:NN \l_@@_end_int \c@jCol
    \keys_set_known:nnN { nicematrix / Rules } { #1 } \l_@@_other_keys_tl
    \@@_hline_i:
    \group_end:
  }
\cs_new_protected:Npn \@@_hline_i:
  {
```

`\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`.

```
    \tl_set:No \l_tmpa_tl { \int_use:N \l_@@_position_int }
    \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
      \l_tmpb_tl
      {
```

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.

```
            \bool_gset_true:N \g_tmpa_bool
```

We test whether we are in a block.

```
          \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
            { \@@_test_hline_in_block:nnnnn ##1 }
            \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
              { \@@_test_hline_in_block:nnnnn ##1 }
            \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
              { \@@_test_hline_in_stroken_block:nnnn ##1 }
            \clist_if_empty:NF \l_@@_corners_clist { \@@_test_in_corner_h: }
            \bool_if:NTF \g_tmpa_bool
              {
                \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.

```
                  { \int_set:Nn \l_@@_local_start_int \l_tmpb_tl }
              }
              {
                \int_compare:nNnT { \l_@@_local_start_int } > { \c_zero_int }
                  {
                    \int_set:Nn \l_@@_local_end_int { \l_tmpb_tl - 1 }
                    \@@_hline_ii:
                    \int_zero:N \l_@@_local_start_int
                  }
              }
        }
      \int_compare:nNnT { \l_@@_local_start_int } > { \c_zero_int }
        {
          \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
          \@@_hline_ii:
        }
    }


\cs_new_protected:Npn \@@_test_in_corner_h:
   {
     \int_compare:nNnTF { \l_tmpa_tl } = { \c@iRow + 1 }
       {
         \@@_if_in_corner:nT { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
           { \bool_set_false:N \g_tmpa_bool }
       }
       {
         \@@_if_in_corner:nT { \l_tmpa_tl - \l_tmpb_tl }
           {
             \int_compare:nNnTF { \l_tmpa_tl } = { \c_one_int }
               { \bool_set_false:N \g_tmpa_bool }
               {
                 \@@_if_in_corner:nT
                   { \int_eval:n { \l_tmpa_tl - 1 } - \l_tmpb_tl }
                   { \bool_set_false:N \g_tmpa_bool }
               }
           }
       }
   }


\cs_new_protected:Npn \@@_hline_ii:
  {
    \tl_clear:N \l_@@_tikz_rule_tl
    \keys_set:no { nicematrix / RulesBis } \l_@@_other_keys_tl
    \bool_if:NTF \l_@@_dotted_bool
      { \@@_hline_iv: }
      {
        \tl_if_empty:NTF \l_@@_tikz_rule_tl
          { \@@_hline_iii: }
          { \@@_hline_v: }
      }
  }
```

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

```
\cs_new_protected:Npn \@@_hline_iii:
  {
    \pgfpicture
    \pgfrememberpicturepositiononpagetrue
    \pgf@relevantforpicturesizefalse
    \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
    \dim_set_eq:NN \l_tmpa_dim \pgf@x
    \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
    \dim_set:Nn \l_tmpb_dim
      {
        \pgf@y
        - 0.5 \l_@@_rule_width_dim
        +
        ( \arrayrulewidth * \l_@@_multiplicity_int
           + \doublerulesep * ( \l_@@_multiplicity_int - 1 ) ) / 2
      }
    \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
    \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
    \bool_lazy_all:nT
      {
        { \int_compare_p:nNn { \l_@@_multiplicity_int } > { \c_one_int } }
        { \cs_if_exist_p:N \CT@drsc@ }
        { ! \tl_if_blank_p:o \CT@drsc@ }
      }
      {
        \group_begin:
        \CT@drsc@
        \dim_set:Nn \l_@@_tmpd_dim
          {
            \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
            * ( \l_@@_multiplicity_int - 1 )
          }
        \pgfpathrectanglecorners
          { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
          { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
        \pgfusepathqfill
        \group_end:
      }
    \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
    \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
    \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
      {
        \dim_sub:Nn \l_tmpb_dim { \arrayrulewidth + \doublerulesep }
        \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
        \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
      }
    \CT@arc@
    \pgfsetlinewidth { 1.1 \arrayrulewidth }
    \pgfsetrectcap
    \pgfusepathqstroke
    \endpgfpicture
  }
```

The following code is for the case of a dotted rule (with our system of rounded dots). The aim is that, by standard the dotted line fits between square brackets (`\hline` doesn't).

```
\begin{bNiceMatrix}
1 & 2 & 3 & 4 \\
\hline
1 & 2 & 3 & 4 \\
\hdottedline
1 & 2 & 3 & 4
\end{bNiceMatrix}
```

$$\left[\begin{array}{cccc} 1 & 2 & 3 & 4 \\ \hline 1 & 2 & 3 & 4 \\ \hdashline 1 & 2 & 3 & 4 \end{array}\right]$$

But, if the user uses `margin`, the dotted line extends to have the same width as a `\hline`.

```
\begin{bNiceMatrix}[margin]
1 & 2 & 3 & 4 \\
\hline
1 & 2 & 3 & 4 \\
\hdottedline
1 & 2 & 3 & 4
\end{bNiceMatrix}
```

$$\left[\begin{array}{cccc} 1 & 2 & 3 & 4 \\ \hline 1 & 2 & 3 & 4 \\ \hdashline 1 & 2 & 3 & 4 \end{array}\right]$$

```
\cs_new_protected:Npn \@@_hline_iv:
  {
    \pgfpicture
    \pgfrememberpicturepositiononpagetrue
    \pgf@relevantforpicturesizefalse
    \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
    \dim_set:Nn \l_@@_y_initial_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
    \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
    \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
    \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
    \int_compare:nNnT { \l_@@_local_start_int } = { \c_one_int }
      {
        \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
        \bool_if:NF \g_@@_delims_bool
          { \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.

```
        \tl_if_eq:NnF \g_@@_left_delim_tl (
          { \dim_add:Nn \l_@@_x_initial_dim  { 0.5 \l_@@_xdots_inter_dim } }
      }
    \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
    \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
    \int_compare:nNnT { \l_@@_local_end_int } = { \c@jCol }
      {
        \dim_add:Nn \l_@@_x_final_dim \l_@@_right_margin_dim
        \bool_if:NF \g_@@_delims_bool
          { \dim_add:Nn \l_@@_x_final_dim \arraycolsep }
        \tl_if_eq:NnF \g_@@_right_delim_tl )
          { \dim_gsub:Nn \l_@@_x_final_dim { 0.5 \l_@@_xdots_inter_dim } }
      }
    \CT@arc@
    \@@_draw_line:
    \endpgfpicture
  }
```

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`).

```
\cs_new_protected:Npn \@@_hline_v:
  {
    \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.

```
    \CT@arc@
    \tl_if_empty:NF \l_@@_rule_color_tl
      { \tl_put_right:Ne \l_@@_tikz_rule_tl { , color = \l_@@_rule_color_tl } }
    \pgfrememberpicturepositiononpagetrue
    \pgf@relevantforpicturesizefalse
    \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
    \dim_set_eq:NN \l_tmpa_dim \pgf@x
    \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
    \dim_set:Nn \l_tmpb_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }
    \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
    \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
    \exp_args:No \tikzset \l_@@_tikz_rule_tl
    \use:e { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
      ( \l_tmpa_dim , \l_tmpb_dim ) --
      ( \l_@@_tmpc_dim , \l_tmpb_dim ) ;
    \end { tikzpicture }
  }
```

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).

```
\cs_new_protected:Npn \@@_draw_hlines:
  {
    \int_step_inline:nnn
      { \bool_lazy_or:nnTF \g_@@_delims_bool \l_@@_except_borders_bool 2 1 }
      {
        \bool_lazy_or:nnTF { \g_@@_delims_bool } { \l_@@_except_borders_bool }
          { \c@iRow }
          { \int_eval:n { \c@iRow + 1 } }
      }
      {
        \str_if_eq:eeF { \l_@@_hlines_clist } { all }
          { \clist_if_in:NnT \l_@@_hlines_clist { ##1 } }
          { \@@_hline:n { position = ##1 , total-width = \arrayrulewidth } }
      }
  }
```

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

```
\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.

```
\cs_set:Npn \@@_Hline_i:n #1
  {
    \peek_remove_spaces:n
      {
        \peek_meaning:NTF \Hline
          { \@@_Hline_ii:nn { #1 + 1 } }
          { \@@_Hline_iii:n { #1 } }
      }
  }
\cs_set:Npn \@@_Hline_ii:nn #1 #2 { \@@_Hline_i:n { #1 } }
\cs_set:Npn \@@_Hline_iii:n #1
  { \@@_collect_options:n { \@@_Hline_iv:nn { #1 } } }
\cs_set_protected:Npn \@@_Hline_iv:nn #1 #2
  {
    \@@_compute_rule_width:n { multiplicity = #1 , #2 }
    \skip_vertical:N \l_@@_rule_width_dim
    \tl_gput_right:Ne \g_@@_pre_code_after_tl
```

```
      {
        \@@_hline:n
          {
            multiplicity = #1 ,
            position = \int_eval:n { \c@iRow + 1 } ,
            total-width = \dim_use:N \l_@@_rule_width_dim ,
            #2
          }
      }
    \egroup
  }
```

### 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`).

```
\cs_new_protected:Npn \@@_custom_line:n #1
  {
    \str_clear_new:N \l_@@_command_str
    \str_clear_new:N \l_@@_ccommand_str
    \str_clear_new:N \l_@@_letter_str
    \tl_clear_new:N \l_@@_other_keys_tl
    \keys_set_known:nnN { nicematrix / custom-line } { #1 } \l_@@_other_keys_tl
    \str_if_eq:eeT \c_backslash_str { \str_head:N \l_@@_command_str }
      {
        \str_set:Ne \l_@@_command_str { \str_tail:N \l_@@_command_str }
```

We delete the last character which is a space.

```
        \str_set:Ne \l_@@_command_str
          { \str_range:Nnn \l_@@_command_str { 1 } { -2 } }
      }
    \str_if_eq:eeT \c_backslash_str { \str_head:N \l_@@_ccommand_str }
      {
        \str_set:Ne \l_@@_ccommand_str
          { \str_tail:N \l_@@_ccommand_str }
        \str_set:Ne \l_@@_ccommand_str
          { \str_range:Nnn \l_@@_ccommand_str { 1 } { -2 } }
      }
```

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.

```
    \bool_lazy_all:nTF
      {
        { \str_if_empty_p:N \l_@@_letter_str }
        { \str_if_empty_p:N \l_@@_command_str }
        { \str_if_empty_p:N \l_@@_ccommand_str }
      }
      { \@@_error:n { No~letter~and~no~command } }
      { \@@_custom_line_i:o \l_@@_other_keys_tl }
  }
\keys_define:nn { nicematrix / custom-line }
  {
    letter .str_set:N = \l_@@_letter_str ,
    letter .value_required:n = true ,
    command .str_set:N = \l_@@_command_str ,
    command .value_required:n = true ,
    ccommand .str_set:N = \l_@@_ccommand_str ,
    ccommand .value_required:n = true ,
  }
```

```
\cs_new_protected:Npn \@@_custom_line_i:n #1
  {
```

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

```
    \bool_set_false:N \l_@@_tikz_rule_bool
    \bool_set_false:N \l_@@_dotted_rule_bool
    \bool_set_false:N \l_@@_color_bool
    \keys_set:nn { nicematrix / custom-line-bis } { #1 }
    \bool_if:NT \l_@@_tikz_rule_bool
      {
        \IfPackageLoadedF { tikz }
          { \@@_error:n { tikz~in~custom-line~without~tikz } }
        \bool_if:NT \l_@@_color_bool
          { \@@_error:n { color~in~custom-line~with~tikz } }
      }
    \bool_if:NT \l_@@_dotted_rule_bool
      {
        \int_compare:nNnT { \l_@@_multiplicity_int } > { \c_one_int }
          { \@@_error:n { key~multiplicity~with~dotted } }
      }
    \str_if_empty:NF \l_@@_letter_str
      {
        \int_compare:nTF { \str_count:N \l_@@_letter_str != 1 }
          { \@@_error:n { Several~letters } }
          {
            \tl_if_in:NoTF
              \c_@@_forbidden_letters_str
              \l_@@_letter_str
              { \@@_error:ne { Forbidden~letter } \l_@@_letter_str }
              {
```

During the analysis 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.

```
                \cs_set_nopar:cpn { @@ _ \l_@@_letter_str : } ##1
                  { \@@_v_custom_line:nn { #1 } }
              }
          }
      }
    \str_if_empty:NF \l_@@_command_str { \@@_h_custom_line:n { #1 } }
    \str_if_empty:NF \l_@@_ccommand_str { \@@_c_custom_line:n { #1 } }
  }
\cs_generate_variant:Nn \@@_custom_line_i:n { o }
\tl_const:Nn \c_@@_forbidden_letters_tl { lcrpmbVX|()[]!@<> }
\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.

```
\keys_define:nn { nicematrix / custom-line-bis }
  {
    multiplicity .int_set:N = \l_@@_multiplicity_int ,
    multiplicity .initial:n = 1 ,
    multiplicity .value_required:n = true ,
    color .code:n = \bool_set_true:N \l_@@_color_bool ,
    color .value_required:n = true ,
    tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
    tikz .value_required:n = true ,
    dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
    dotted .value_forbidden:n = true ,
    total-width .code:n = { } ,
    total-width .value_required:n = true ,
    width .code:n = { } ,
    width .value_required:n = true ,
    sep-color .code:n = { } ,
    sep-color .value_required:n = true ,
    unknown .code:n =
      \@@_unknown_key:nn
        { nicematrix / custom-line-bis }
        { Unknown~key~for~custom-line }
  }
```

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

```
\bool_new:N \l_@@_dotted_rule_bool
\bool_new:N \l_@@_tikz_rule_bool
\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`.

```
\keys_define:nn { nicematrix / custom-line-width }
  {
    multiplicity .int_set:N = \l_@@_multiplicity_int ,
    multiplicity .initial:n = 1 ,
    multiplicity .value_required:n = true ,
    tikz .code:n = \bool_set_true:N \l_@@_tikz_rule_bool ,
    total-width .code:n = \dim_set:Nn \l_@@_rule_width_dim { #1 }
                          \bool_set_true:N \l_@@_total_width_bool ,
    total-width .value_required:n = true ,
    width .meta:n = { total-width = #1 } ,
    dotted .code:n = \bool_set_true:N \l_@@_dotted_rule_bool ,
  }
```

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`).

```
\cs_new_protected:Npn \@@_h_custom_line:n #1
  {
```

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`).

```
    \cs_set_nopar:cpn { nicematrix - \l_@@_command_str } { \Hline [ #1 ] }
    \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_command_str
  }
```

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`).

```
\cs_new_protected:Npn \@@_c_custom_line:n #1
  {
```

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

```
    \exp_args:Nc \NewExpandableDocumentCommand
      { nicematrix - \l_@@_ccommand_str }
      { O { } m }
      {
        \noalign
          {
            \@@_compute_rule_width:n { #1 , ##1 }
            \skip_vertical:n { \l_@@_rule_width_dim }
            \clist_map_inline:nn
              { ##2 }
              { \@@_c_custom_line_i:nn { #1 , ##1 } { ####1 } }
          }
      }
    \seq_put_left:No \l_@@_custom_line_commands_seq \l_@@_ccommand_str
  }
```

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*.

```
\cs_new_protected:Npn \@@_c_custom_line_i:nn #1 #2
  {
    \tl_if_in:nnTF { #2 } { - }
      { \@@_cut_on_hyphen:w #2 \q_stop }
      { \@@_cut_on_hyphen:w #2 - #2 \q_stop }
    \tl_gput_right:Ne \g_@@_pre_code_after_tl
      {
        \@@_hline:n
          {
            #1 ,
            start = \l_tmpa_tl ,
            end = \l_tmpb_tl ,
            position = \int_eval:n { \c@iRow + 1 } ,
            total-width = \dim_use:N \l_@@_rule_width_dim
          }
      }
  }
```

```
\cs_new_protected:Npn \@@_compute_rule_width:n #1
  {
    \bool_set_false:N \l_@@_tikz_rule_bool
    \bool_set_false:N \l_@@_total_width_bool
    \bool_set_false:N \l_@@_dotted_rule_bool
    \keys_set_known:nn { nicematrix / custom-line-width } { #1 }
    \bool_if:NF \l_@@_total_width_bool
      {
        \bool_if:NTF \l_@@_dotted_rule_bool
          { \dim_set:Nn \l_@@_rule_width_dim { 2 \l_@@_xdots_radius_dim } }
          {
            \bool_if:NF \l_@@_tikz_rule_bool
              {
                \dim_set:Nn \l_@@_rule_width_dim
                  {
                    \arrayrulewidth * \l_@@_multiplicity_int
                    + \doublerulesep * ( \l_@@_multiplicity_int - 1 )
                  }
              }
          }
      }
  }
```

The following constructions aims to allow cumulative blocks of options between square brackets such as in `I[color=blue][tikz=dashed]`.

```
\cs_new_protected:Npn \@@_v_custom_line:nn #1 #2
  {
    \str_if_eq:nnTF { #2 } { [ }
      { \@@_v_custom_line_i:nw { #1 } [ }
      { \@@_v_custom_line_ii:nn { #2 } { #1 } }
  }
\cs_new_protected:Npn \@@_v_custom_line_i:nw #1 [ #2 ]
  { \@@_v_custom_line:nn { #1 , #2 } }
```

```
\cs_new_protected:Npn \@@_v_custom_line_ii:nn #1 #2
  {
    \@@_compute_rule_width:n { #2 }
```

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

```
    \tl_gput_right:Ne \g_@@_array_preamble_tl
      { \exp_not:N ! { \skip_horizontal:n { \dim_use:N \l_@@_rule_width_dim } } }
    \tl_gput_right:Ne \g_@@_pre_code_after_tl
      {
        \@@_vline:n
          {
            #2 ,
            position = \int_eval:n { \c@jCol + 1 } ,
            total-width = \dim_use:N \l_@@_rule_width_dim
          }
      }
    \@@_rec_preamble:n #1
  }
```

```
\@@_custom_line:n
  { 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`.

```
\cs_new_protected:Npn \@@_test_hline_in_block:nnnnn #1 #2 #3 #4 #5
  {
    \int_compare:nNnT { \l_tmpa_tl } > { #1 }
      {
        \int_compare:nNnT { \l_tmpa_tl } < { #3 + 1 }
          {
            \int_compare:nNnT { \l_tmpb_tl } > { #2 - 1 }
              {
                \int_compare:nNnT { \l_tmpb_tl } < { #4 + 1 }
                  { \bool_gset_false:N \g_tmpa_bool }
              }
          }
      }
  }
```

The same for vertical rules.

```
\cs_new_protected:Npn \@@_test_vline_in_block:nnnnn #1 #2 #3 #4 #5
  {
    \int_compare:nNnT { \l_tmpa_tl } > { #1 - 1 }
      {
        \int_compare:nNnT { \l_tmpa_tl } < { #3 + 1 }
          {
            \int_compare:nNnT { \l_tmpb_tl } > { #2 }
              {
                \int_compare:nNnT { \l_tmpb_tl } < { #4 + 1 }
                  { \bool_gset_false:N \g_tmpa_bool }
              }
          }
      }
  }
```

```
\cs_new_protected:Npn \@@_test_hline_in_stroken_block:nnnn #1 #2 #3 #4
  {
    \int_compare:nNnT { \l_tmpb_tl } > { #2 - 1 }
      {
        \int_compare:nNnT { \l_tmpb_tl } < { #4 + 1 }
          {
            \int_compare:nNnTF { \l_tmpa_tl } = { #1 }
              { \bool_gset_false:N \g_tmpa_bool }
              {
                \int_compare:nNnT { \l_tmpa_tl } = { #3 + 1 }
                  { \bool_gset_false:N \g_tmpa_bool }
              }
          }
      }
  }
\cs_new_protected:Npn \@@_test_vline_in_stroken_block:nnnn #1 #2 #3 #4
  {
    \int_compare:nNnT { \l_tmpa_tl } > { #1 - 1 }
      {
        \int_compare:nNnT { \l_tmpa_tl } < { #3 + 1 }
          {
            \int_compare:nNnTF { \l_tmpb_tl } = { #2 }
              { \bool_gset_false:N \g_tmpa_bool }
              {
                \int_compare:nNnT { \l_tmpb_tl } = { #4 + 1 }
                  { \bool_gset_false:N \g_tmpa_bool }
              }
          }
      }
  }
```

## 23 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.

```
\cs_new_protected:Npn \@@_compute_corners:
  {
    \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
      { \@@_mark_cells_of_block:nnnnn ##1 }
```

The list `\l_@@_corners_cells_clist` will be the list of all the empty cells (and not in a block) considered in the corners of the array. We use a `clist` instead of a `seq` because we will frequently search in that list (and searching in a `clist` is faster than searching in a `seq`).

```
    \clist_clear:N \l_@@_corners_cells_clist
    \clist_map_inline:Nn \l_@@_corners_clist
      {
        \str_case:nnF { ##1 }
          {
            { NW }
            { \@@_compute_a_corner:nnnnnn 1 1 1 1 \c@iRow \c@jCol }
            { NE }
            { \@@_compute_a_corner:nnnnnn 1 \c@jCol 1 { -1 } \c@iRow 1 }
            { SW }
            { \@@_compute_a_corner:nnnnnn \c@iRow 1 { -1 } 1 1 \c@jCol }
            { SE }
            { \@@_compute_a_corner:nnnnnn \c@iRow \c@jCol { -1 } { -1 } 1 1 }
          }
          { \@@_error:nn { bad~corner } { ##1 } }
      }
```

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

```
    \clist_if_empty:NF \l_@@_corners_cells_clist
      {
```

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 colors the `rows`, `columns` and `cells` must not color the cells in the corners.

```
        \tl_gput_right:Ne \g_@@_aux_tl
          {
            \clist_set:Nn \exp_not:N \l_@@_corners_cells_clist
              { \l_@@_corners_cells_clist }
          }
      }
  }
```

```
\cs_new_protected:Npn \@@_mark_cells_of_block:nnnnn #1 #2 #3 #4 #5
  {
    \int_step_inline:nnn { #1 } { #3 }
      {
        \int_step_inline:nnn { #2 } { #4 }
          { \cs_set_nopar:cpn { @@ _ block _ ##1 - ####1 } { } }
      }
  }
```

```
\prg_new_conditional:Npnn \@@_if_in_block:nn #1 #2 { p }
  {
    \cs_if_exist:cTF
      { @@ _ block _ \int_eval:n { #1 } - \int_eval:n { #2 } }
      { \prg_return_true: }
      { \prg_return_false: }
  }
```

"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_clist`.

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.

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

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.

```
    \bool_set_false:N \l_tmpa_bool
    \int_zero_new:N \l_@@_last_empty_row_int
    \int_set:Nn \l_@@_last_empty_row_int { #1 }
    \int_step_inline:nnnn { #1 } { #3 } { #5 }
      {
        \bool_lazy_or:nnTF
          {
            \cs_if_exist_p:c
              { pgf @ sh @ ns @ \@@_env: - ##1 - \int_eval:n { #2 } }
          }
          { \@@_if_in_block_p:nn { ##1 } { #2 } }
          { \bool_set_true:N \l_tmpa_bool }
          {
            \bool_if:NF \l_tmpa_bool
              { \int_set:Nn \l_@@_last_empty_row_int { ##1 } }
          }
      }
```

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

```
    \bool_set_false:N \l_tmpa_bool
    \int_zero_new:N \l_@@_last_empty_column_int
    \int_set:Nn \l_@@_last_empty_column_int { #2 }
    \int_step_inline:nnnn { #2 } { #4 } { #6 }
      {
        \bool_lazy_or:nnTF
          {
            \cs_if_exist_p:c
              { pgf @ sh @ ns @ \@@_env: - \int_eval:n { #1 } - ##1 }
          }
          { \@@_if_in_block_p:nn { #1 } { ##1 } }
          { \bool_set_true:N \l_tmpa_bool }
          {
            \bool_if:NF \l_tmpa_bool
              { \int_set:Nn \l_@@_last_empty_column_int { ##1 } }
          }
      }
```

Now, we loop over the rows.

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

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

```
        \bool_set_false:N \l_tmpa_bool
        \int_step_inline:nnnn { #2 } { #4 } { \l_@@_last_empty_column_int }
          {
            \bool_lazy_or:nnTF
              { \cs_if_exist_p:c { pgf @ sh @ ns @ \@@_env: - ##1 - ####1 } }
              { \@@_if_in_block_p:nn  { ##1 } { ####1 } }
              { \bool_set_true:N \l_tmpa_bool }
              {
                \bool_if:NF \l_tmpa_bool
                  {
                    \int_set:Nn \l_@@_last_empty_column_int { ####1 }
                    \clist_put_right:Nn
                      \l_@@_corners_cells_clist
                      { ##1 - ####1 }
                    \cs_set_nopar:cpn { @@ _ corner _ ##1 - ####1 } { }
                  }
              }
          }
      }
  }
```

Of course, instead of the following lines, we could have use `\prg_new_conditional:Npnn`.

```
\cs_new:Npn \@@_if_in_corner:nT #1 { \cs_if_exist:cT { @@ _ corner _ #1 } }
\cs_new:Npn \@@_if_in_corner:nF #1 { \cs_if_exist:cF { @@ _ corner _ #1 } }
```

Instead of the previous lines, we could have used `\l_@@_corners_cells_clist` but it's less efficient: `\clist_if_in:NeT \l_@@_corners_cells_clist { #1 } ...`

## 24 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.

```
\bool_new:N \l_@@_block_auto_columns_width_bool
```

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

```
\keys_define:nn { nicematrix / NiceMatrixBlock }
  {
    auto-columns-width .code:n =
      {
        \bool_set_true:N \l_@@_block_auto_columns_width_bool
        \dim_gzero_new:N \g_@@_max_cell_width_dim
        \bool_set_true:N \l_@@_auto_columns_width_bool
      }
  }
```

```
\NewDocumentEnvironment { NiceMatrixBlock } { ! O { } }
  {
    \int_gincr:N \g_@@_NiceMatrixBlock_int
    \dim_zero:N \l_@@_columns_width_dim
    \keys_set:nn { nicematrix / NiceMatrixBlock } { #1 }
    \bool_if:NT \l_@@_block_auto_columns_width_bool
      {
        \cs_if_exist:cT
          { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
          {
            \dim_set:Nn \l_@@_columns_width_dim
              {
                \use:c
                  { @@_max_cell_width _ \int_use:N \g_@@_NiceMatrixBlock_int }
              }
          }
      }
  }
```

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).

```
  {
    \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.

```
      { \int_gdecr:N \g_@@_NiceMatrixBlock_int }
      {
        \bool_if:NT \l_@@_block_auto_columns_width_bool
          {
            \iow_shipout:Nn \@mainaux \ExplSyntaxOn
            \iow_shipout:Ne \@mainaux
              {
                \cs_gset:cpn
                  { @@ _ 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.

```
                  { \dim_eval:n { \g_@@_max_cell_width_dim + \arrayrulewidth } }
              }
            \iow_shipout:Nn \@mainaux \ExplSyntaxOff
          }
      }
    \ignorespacesafterend
  }
```

## 25 The extra nodes

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).

```
\cs_new_protected:Npn \@@_create_extra_nodes:
  {
    \bool_if:nTF \l_@@_medium_nodes_bool
      {
        \bool_if:NTF \l_@@_no_cell_nodes_bool
          { \@@_error:n { extra-nodes~with~no-cell-nodes } }
          {
            \bool_if:NTF \l_@@_large_nodes_bool
              \@@_create_medium_and_large_nodes:
              \@@_create_medium_nodes:
          }
      }
      {
        \bool_if:NT \l_@@_large_nodes_bool
          {
            \bool_if:NTF \l_@@_no_cell_nodes_bool
              { \@@_error:n { extra-nodes~with~no-cell-nodes } }
              \@@_create_large_nodes:
          }
      }
  }
```

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`.

```
\cs_new_protected:Npn \@@_computations_for_medium_nodes:
  {
    \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
      {
        \dim_zero_new:c { l_@@_row_ \@@_i: _min_dim }
        \dim_set_eq:cN { l_@@_row_ \@@_i: _min_dim } \c_max_dim
        \dim_zero_new:c { l_@@_row_ \@@_i: _max_dim }
        \dim_set:cn { l_@@_row_ \@@_i: _max_dim } { - \c_max_dim }
      }
    \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
      {
        \dim_zero_new:c { l_@@_column_ \@@_j: _min_dim }
        \dim_set_eq:cN { l_@@_column_ \@@_j: _min_dim } \c_max_dim
        \dim_zero_new:c { l_@@_column_ \@@_j: _max_dim }
        \dim_set:cn { l_@@_column_ \@@_j: _max_dim } { - \c_max_dim }
      }
```

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

```
    \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
      {
        \int_step_variable:nnNn
          \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.

```
          {
            \cs_if_exist:cT
              { 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`.

```
              {
                \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { south~west }
                \dim_set:cn { l_@@_row_ \@@_i: _min_dim }
                  { \dim_min:vn { l_@@_row _ \@@_i: _min_dim } \pgf@y }
                \seq_if_in:NeF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
                  {
                    \dim_set:cn { l_@@_column _ \@@_j: _min_dim }
                      { \dim_min:vn { l_@@_column _ \@@_j: _min_dim } \pgf@x }
                  }
```

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`.

```
                \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { north~east }
                \dim_set:cn { l_@@_row _ \@@_i: _ max_dim }
                  { \dim_max:vn { l_@@_row _ \@@_i: _ max_dim } { \pgf@y } }
                \seq_if_in:NeF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
                  {
                    \dim_set:cn { l_@@_column _ \@@_j: _ max_dim }
                      { \dim_max:vn { l_@@_column _ \@@_j: _max_dim } { \pgf@x } }
                  }
              }
          }
      }
```

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

```
    \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
      {
        \dim_compare:nNnT
          { \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } } = \c_max_dim
          {
            \@@_qpoint:n { row - \@@_i: - base }
            \dim_set:cn { l_@@_row _ \@@_i: _ max _ dim } \pgf@y
            \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim } \pgf@y
          }
      }
    \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
      {
        \dim_compare:nNnT
          { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } } = \c_max_dim
          {
            \@@_qpoint:n { col - \@@_j: }
            \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim } \pgf@y
            \dim_set:cn { l_@@_column _ \@@_j: _ min _ dim } \pgf@y
          }
      }
  }
```

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

```
\cs_new_protected:Npn \@@_create_medium_nodes:
  {
    \pgfpicture
    \pgfrememberpicturepositiononpagetrue
    \pgf@relevantforpicturesizefalse
    \@@_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".

```
    \tl_set:Nn \l_@@_suffix_tl { -medium }
    \@@_create_nodes:
    \endpgfpicture
  }
```

The command `\@@_create_large_nodes:` must be used when we want to create only the "large nodes" and not the medium ones[15]. 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:`.

```
\cs_new_protected:Npn \@@_create_large_nodes:
  {
    \pgfpicture
      \pgfrememberpicturepositiononpagetrue
      \pgf@relevantforpicturesizefalse
      \@@_computations_for_medium_nodes:
      \@@_computations_for_large_nodes:
      \tl_set:Nn \l_@@_suffix_tl { - large }
      \@@_create_nodes:
    \endpgfpicture
  }
\cs_new_protected:Npn \@@_create_medium_and_large_nodes:
  {
    \pgfpicture
      \pgfrememberpicturepositiononpagetrue
      \pgf@relevantforpicturesizefalse
      \@@_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".

```
      \tl_set:Nn \l_@@_suffix_tl { - medium }
      \@@_create_nodes:
      \@@_computations_for_large_nodes:
      \tl_set:Nn \l_@@_suffix_tl { - large }
      \@@_create_nodes:
    \endpgfpicture
  }
```

For "large nodes", the exterior rows and columns don't interfere. That's why the loop over the columns will start at 1 and stop at `\c@jCol` (and not `\g_@@_col_total_int`). Idem for the rows.

```
\cs_new_protected:Npn \@@_computations_for_large_nodes:
  {
    \int_set_eq:NN \l_@@_first_row_int \c_one_int
    \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`.

```
    \int_step_variable:nNn { \c@iRow - 1 } \@@_i:
      {
        \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim }
```

[15]If we want to create both, we have to use `\@@_create_medium_and_large_nodes:`

```
          {
            (
              \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } +
              \dim_use:c { l_@@_row _ \int_eval:n { \@@_i: + 1 }  _ max _ dim }
            )
            / 2
          }
        \dim_set_eq:cc { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim }
          { l_@@_row_ \@@_i: _min_dim }
      }
    \int_step_variable:nNn { \c@jCol - 1 } \@@_j:
      {
        \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim }
          {
            (
              \dim_use:c { l_@@_column _ \@@_j: _ max _ dim } +
              \dim_use:c
                { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
            )
            / 2
          }
        \dim_set_eq:cc { l_@@_column _ \int_eval:n { \@@_j: + 1 } _ min _ dim }
          { l_@@_column _ \@@_j: _ max _ dim }
      }
```

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

```
    \dim_sub:cn
      { l_@@_column _ 1 _ min _ dim }
      \l_@@_left_margin_dim
    \dim_add:cn
      { l_@@_column _ \int_use:N \c@jCol _ max _ dim }
      \l_@@_right_margin_dim
  }
```

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`).

```
\cs_new_protected:Npn \@@_create_nodes:
  {
    \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
      {
        \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
          {
```

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

```
            \@@_pgf_rect_node:nnnnn
              { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
              { \dim_use:c { l_@@_column_ \@@_j: _min_dim } }
              { \dim_use:c { l_@@_row_ \@@_i: _min_dim } }
              { \dim_use:c { l_@@_column_ \@@_j: _max_dim } }
              { \dim_use:c { l_@@_row_ \@@_i: _max_dim } }
            \str_if_empty:NF \l_@@_name_str
              {
                \pgfnodealias
                  { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
                  { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
              }
          }
      }
    \int_step_inline:nn { \c@iRow }
```

```
      {
        \pgfnodealias
          { \@@_env: - ##1 - last \l_@@_suffix_tl }
          { \@@_env: - ##1 - \int_use:N \c@jCol \l_@@_suffix_tl }
      }
    \int_step_inline:nn { \c@jCol }
      {
        \pgfnodealias
          { \@@_env: - last - ##1 \l_@@_suffix_tl }
          { \@@_env: - \int_use:N \c@iRow - ##1 \l_@@_suffix_tl }
      }
    \pgfnodealias % added 2025-04-05
      { \@@_env: - last - last \l_@@_suffix_tl }
      { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol \l_@@_suffix_tl }
```

Now, we create the nodes for the cells of the `\multicolumn`. We recall that we have stored in `\g_@@_multicolumn_cells_seq` the list of the cells where a `\multicolumn{`$n$`}{...}{...}` with $n$>1 was issued and in `\g_@@_multicolumn_sizes_seq` the correspondent values of $n$.

```
      \seq_map_pairwise_function:NNN
      \g_@@_multicolumn_cells_seq
      \g_@@_multicolumn_sizes_seq
      \@@_node_for_multicolumn:nn
  }
```

```
\cs_new_protected:Npn \@@_extract_coords_values: #1 - #2 \q_stop
  {
    \cs_set_nopar:Npn \@@_i: { #1 }
    \cs_set_nopar:Npn \@@_j: { #2 }
  }
```

The command `\@@_node_for_multicolumn:nn` takes two arguments. The first is the position of the cell where the command `\multicolumn{`$n$`}{...}{...}` was issued in the format $i$-$j$ and the second is the value of $n$ (the length of the "multi-cell").

```
\cs_new_protected:Npn \@@_node_for_multicolumn:nn #1 #2
  {
    \@@_extract_coords_values: #1 \q_stop
    \@@_pgf_rect_node:nnnnn
      { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
      { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } }
      { \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } }
      { \dim_use:c { l_@@_column _ \int_eval:n { \@@_j: +#2-1 } _ max _ dim } }
      { \dim_use:c { l_@@_row _ \@@_i: _ max _ dim } }
    \str_if_empty:NF \l_@@_name_str
      {
        \pgfnodealias
          { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
          { \int_use:N \g_@@_env_int - \@@_i: - \@@_j: \l_@@_suffix_tl }
      }
  }
```

## 26 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).

```
\keys_define:nn { nicematrix / Block / FirstPass }
  {
    j .code:n = \str_set:Nn \l_@@_hpos_block_str j
                \bool_set_true:N \l_@@_p_block_bool ,
    j .value_forbidden:n = true ,
    l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
    l .value_forbidden:n = true ,
    r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
    r .value_forbidden:n = true ,
    c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
    c .value_forbidden:n = true ,
    L .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
    L .value_forbidden:n = true ,
    R .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
    R .value_forbidden:n = true ,
    C .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
    C .value_forbidden:n = true ,
    t .code:n = \str_set:Nn \l_@@_vpos_block_str t ,
    t .value_forbidden:n = true ,
    T .code:n = \str_set:Nn \l_@@_vpos_block_str T ,
    T .value_forbidden:n = true ,
    b .code:n = \str_set:Nn \l_@@_vpos_block_str b ,
    b .value_forbidden:n = true ,
    B .code:n = \str_set:Nn \l_@@_vpos_block_str B ,
    B .value_forbidden:n = true ,
    m .code:n = \str_set:Nn \l_@@_vpos_block_str c ,
    m .value_forbidden:n = true ,
    v-center .meta:n = m ,
    p .code:n = \bool_set_true:N \l_@@_p_block_bool ,
    p .value_forbidden:n = true ,
    color .code:n =
      \@@_color:n { #1 }
      \tl_set_rescan:Nnn
        \l_@@_draw_tl
        { \char_set_catcode_other:N ! }
        { #1 } ,
    color .value_required:n = true ,
    respect-arraystretch .code:n =
      \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
    respect-arraystretch .value_forbidden:n = true ,
  }
```

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.

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

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

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).

```
    \tl_if_blank:nTF { #2 }
      { \@@_Block_ii:nnnnn \c_one_int \c_one_int }
      {
        \tl_if_in:nnTF { #2 } { - }
          {
            \int_compare:nNnTF { \char_value_catcode:n { 45 } } = { 13 }
            \@@_Block_i_czech:w \@@_Block_i:w
            #2 \q_stop
          }
          {
            \@@_error:nn { Bad~argument~for~Block } { #2 }
            \@@_Block_ii:nnnnn \c_one_int \c_one_int
          }
      }
    { #1 } { #3 } { #4 }
    \ignorespaces
}
```

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

```
\cs_new:Npn \@@_Block_i:w #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`.

```
{
  \char_set_catcode_active:N -
  \cs_new:Npn \@@_Block_i_czech:w #1-#2 \q_stop { \@@_Block_ii:nnnnn { #1 } { #2 } }
}
```

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.

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

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).

```
    \bool_lazy_or:nnTF
      { \tl_if_blank_p:n { #1 } }
      { \str_if_eq_p:ee { * } { #1 } }
      { \int_set:Nn \l_tmpa_int { 100 } }
      { \int_set:Nn \l_tmpa_int { #1 } }
    \bool_lazy_or:nnTF
      { \tl_if_blank_p:n { #2 } }
      { \str_if_eq_p:ee { * } { #2 } }
      { \int_set:Nn \l_tmpb_int { 100 } }
      { \int_set:Nn \l_tmpb_int { #2 } }
```

If the block is mono-column.

```
    \int_compare:nNnTF { \l_tmpb_int } = { \c_one_int }
      {
        \tl_if_empty:NTF \l_@@_hpos_cell_tl
          { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_c_str }
          { \str_set:No \l_@@_hpos_block_str \l_@@_hpos_cell_tl }
      }
      { \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.

```
    \keys_set_known:nn { nicematrix / Block / FirstPass } { #3 }
    \tl_set:Ne \l_tmpa_tl
      {
        { \int_use:N \c@iRow }
        { \int_use:N \c@jCol }
        { \int_eval:n { \c@iRow + \l_tmpa_int - 1 } }
        { \int_eval:n { \c@jCol + \l_tmpb_int - 1 } }
      }
```

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).

```
    \bool_set_false:N \l_tmpa_bool
    \bool_if:NT \l_@@_amp_in_blocks_bool
```

`\tl_if_in:nnT` is slightly faster than `\str_if_in:nnT`.

```
      { \tl_if_in:nnT { #5 } { & } { \bool_set_true:N \l_tmpa_bool } }
    \bool_case:nF
      {
        \l_tmpa_bool                                  { \@@_Block_vii:eennn }
        \l_@@_p_block_bool                            { \@@_Block_vi:eennn }
```

For the blocks mono-column, we will compose right away 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.

```
        \l_@@_X_bool                                  { \@@_Block_v:eennn }
        { \tl_if_empty_p:n { #5 } }                   { \@@_Block_v:eennn }
        { \int_compare_p:nNn \l_tmpa_int = \c_one_int } { \@@_Block_iv:eennn }
        { \int_compare_p:nNn \l_tmpb_int = \c_one_int } { \@@_Block_iv:eennn }
      }
      { \@@_Block_v:eennn }
    { \l_tmpa_int } { \l_tmpb_int } { #3 } { #4 } { #5 }
  }
```

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 away in a box (because we have to take into account the dimensions of that box for the width of the current column or the height and the depth of the current row). However, that box will be put in the array *after the construction of the array* (by using PGF) with `\@@_draw_blocks:` and above all `\@@_Block_v:nnnnnn` which will do the main job.
`#1` is $i$ (the number of rows of the block), `#2` is $j$ (the number of columns of the block), `#3` is the list of *key=values* pairs, `#4` are the tokens to put before the potential math mode and before the composition of the block and `#5` is the label (=content) of the block.

```
\cs_new_protected:Npn \@@_Block_iv:nnnnn #1 #2 #3 #4 #5
  {
    \int_gincr:N \g_@@_block_box_int
    \cs_set_eq:NN \cellcolor \@@_cellcolor_error
    \cs_set_eq:NN \rowcolor \@@_rowcolor_error
    \cs_set_protected_nopar:Npn \diagbox ##1 ##2
      {
        \tl_gput_right:Ne \g_@@_pre_code_after_tl
          {
            \@@_actually_diagbox:nnnnnn
              { \int_use:N \c@iRow }
              { \int_use:N \c@jCol }
              { \int_eval:n { \c@iRow + #1 - 1 } }
              { \int_eval:n { \c@jCol + #2 - 1 } }
              { \g_@@_row_style_tl \exp_not:n { ##1 } }
              { \g_@@_row_style_tl \exp_not:n { ##2 } }
          }
      }
    \box_gclear_new:c
      { 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!

```
    \hbox_gset:cn
      { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
      {
```

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`).

```
        \tl_if_empty:NTF \l_@@_color_tl
          { \int_compare:nNnT { #2 } = { \c_one_int } { \set@color } }
          { \@@_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`.

```
        \int_compare:nNnT { #1 } = { \c_one_int }
          {
            \int_if_zero:nTF { \c@iRow }
              {
```

In the following code, the value of `code-for-first-row` contains a `\Block` (in order to have the “first row” centered). But, that block will be executed, since it is entirely contained in the first row, the value of `code-for-first-row` will be inserted once again... with the same command `\Block`. That’s why we have to nullify the command `\Block`.

```
$\begin{bNiceMatrix}%
  [
    r,
    first-row,
    last-col,
    code-for-first-row = \Block{}{\scriptstyle\color{blue} \arabic{jCol}},
    code-for-last-col = \scriptstyle \color{blue} \arabic{iRow}
  ]
     &   &    &   & \\
  -2 & 3 & -4 & 5 & \\
  3 & -4 & 5 & -6 & \\
  -4 & 5 & -6 & 7 & \\
  5 & -6 & 7 & -8 & \\
\end{bNiceMatrix}$
```

```
                \cs_set_eq:NN \Block \@@_NullBlock:
                \l_@@_code_for_first_row_tl
              }
              {
                \int_compare:nNnT { \c@iRow } = { \l_@@_last_row_int }
                  {
                    \cs_set_eq:NN \Block \@@_NullBlock:
                    \l_@@_code_for_last_row_tl
                  }
              }
            \g_@@_row_style_tl
          }
```

The following command will be no-op when `respect-arraystretch` is in force.

```
        \@@_reset_arraystretch:
        \dim_zero:N \extrarowheight
```

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

```
        #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.).

```
        \@@_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}`.

```
        \bool_if:NTF \l_@@_tabular_bool
          {
            \bool_lazy_all:nTF
              {
                { \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.

```
                {
                  ! \dim_compare_p:nNn
                      { \l_@@_col_width_dim } < { \c_zero_dim }
                }
                { ! \g_@@_rotate_bool }
              }
```

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

```
              {
                \use:e
                  {
```

Curiously, `\exp_not:N` is still mandatory when `tagging=on`.

```
                    \exp_not:N \begin { minipage }
                      [ \str_lowercase:f \l_@@_vpos_block_str ]
                      { \l_@@_col_width_dim }
                     \str_case:on \l_@@_hpos_block_str
                       { c \centering r \raggedleft l \raggedright }
                  }
                  #5
                \end { minipage }
              }
```

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

```
              {
                \use:e
                  {
```

Curiously, `\exp_not:N` is still mandatory when `tagging=on`.

```
                    \exp_not:N \begin { tabular }
                      [ \str_lowercase:f \l_@@_vpos_block_str ]
                      { @ { } \l_@@_hpos_block_str @ { } }
                  }
                  #5
                \end { tabular }
              }
          }
```

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

```
          {
            $ % $
            \use:e
              {
```

Curiously, `\exp_not:N` is still mandatory when `tagging=on`.

```
                  \exp_not:N \begin { array }
                    [ \str_lowercase:f \l_@@_vpos_block_str ]
                    { @ { } \l_@@_hpos_block_str @ { } }
                  }
                  #5
                \end { array }
                $ % $
              }
          }
```

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).

```
        \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.

```
        \int_compare:nNnT { #2 } = { \c_one_int }
          {
            \dim_gset:Nn \g_@@_blocks_wd_dim
              {
                \dim_max:nn
                  { \g_@@_blocks_wd_dim }
                  {
                    \box_wd:c
                      { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
                  }
              }
          }
```

If we are in a mono-row block we take into account the height and the depth of that block for the height and the depth of the row, excepted when the block uses explicitely an option of vertical position `T` or `B`. Remind that if the user has not used a key for the vertical position of the block, then `\l_@@_vpos_block_str` remains empty.

```
        \int_compare:nNnT { #1 } = { \c_one_int }
          {
            \bool_lazy_any:nT
              {
                { \str_if_empty_p:N \l_@@_vpos_block_str }
                { \str_if_eq_p:ee { \l_@@_vpos_block_str } { t } }
                { \str_if_eq_p:ee { \l_@@_vpos_block_str } { b } }
              }
              { \@@_adjust_blocks_ht_dp: }
          }
        \seq_gput_right:Ne \g_@@_blocks_seq
          {
            \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.

```
            {
              \exp_not:n { #3 } ,
              \l_@@_hpos_block_str ,
```

Now, we put a key for the vertical alignment.

```
              \bool_if:NT \g_@@_rotate_bool
                {
                  \bool_if:NTF \g_@@_rotate_c_bool
                    { m }
                    {
```

```
                    \int_compare:nNnT { \c@iRow } = { \l_@@_last_row_int }
                      { T }
                  }
              }
          }
          {
            \box_use_drop:c
              { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
          }
        }
      \bool_set_false:N \g_@@_rotate_c_bool
    }
\cs_new_protected:Npn \@@_adjust_blocks_ht_dp:
    {
      \dim_gset:Nn \g_@@_blocks_ht_dim
        {
          \dim_max:nn
            { \g_@@_blocks_ht_dim }
            {
              \box_ht:c
                { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
            }
        }
      \dim_gset:Nn \g_@@_blocks_dp_dim
        {
          \dim_max:nn
            { \g_@@_blocks_dp_dim }
            {
              \box_dp:c
                { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
            }
        }
    }


\cs_new:Npn \@@_adjust_hpos_rotate:
    {
      \bool_if:NT \g_@@_rotate_bool
        {
          \str_set:Ne \l_@@_hpos_block_str
            {
              \bool_if:NTF \g_@@_rotate_c_bool
                { c }
                {
                  \str_case:onF \l_@@_vpos_block_str
                    { b l B l t r T r }
                    {
                      \int_compare:nNnTF { \c@iRow } = { \l_@@_last_row_int }
                        { r }
                        { l }
                    }
                }
            }
        }
    }
\cs_generate_variant:Nn \@@_Block_iv:nnnnn { e e }
```

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

```
\cs_new_protected:Npn \@@_rotate_box_of_block:
    {
      \box_grotate:cn
      { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
      { 90 }
    \int_compare:nNnT { \c@iRow } = { \l_@@_last_row_int }
      {
        \vbox_gset_top:cn
          { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
          {
            \skip_vertical:n { 0.8 ex }
            \box_use:c
              { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
          }
      }
    \bool_if:NT \g_@@_rotate_c_bool
      {
        \hbox_gset:cn
          { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
          {
            $ % $
            \vcenter
              {
                \box_use:c
                { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
              }
            $ % $
          }
      }
  }
```

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 away 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.

```
\cs_new_protected:Npn \@@_Block_v:nnnnn #1 #2 #3 #4 #5
  {
    \seq_gput_right:Ne \g_@@_blocks_seq
      {
        \l_tmpa_tl
        { \exp_not:n { #3 } }
        {
          \bool_if:NTF \l_@@_tabular_bool
            {
              \group_begin:
```

The following command will be no-op when `respect-arraystretch` is in force.

```
              \@@_reset_arraystretch:
              \exp_not:n
                {
                  \dim_zero:N \extrarowheight
                  #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).

```
                  \tag_if_active:T { \tag_stop:n { table } }
                  \use:e
                    {
                      \exp_not:N \begin { tabular } [ \l_@@_vpos_block_str ]
                      { @ { } \l_@@_hpos_block_str @ { } }
                    }
                    #5
                  \end { tabular }
                }
              \group_end:
            }
```

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

```
            {
              \group_begin:
```

The following will be no-op when respect-arraystretch is in force.

```
              \@@_reset_arraystretch:
              \exp_not:n
                {
                  \dim_zero:N \extrarowheight
                  #4
                  $ % $
                  \use:e
                    {
                      \exp_not:N \begin { array } [ \l_@@_vpos_block_str ]
                      { @ { } \l_@@_hpos_block_str @ { } }
                    }
                    #5
                  \end { array }
                  $ % $
                }
              \group_end:
            }
        }
    }
  }
\cs_generate_variant:Nn \@@_Block_v:nnnnn { e e }
```

The following macro is for the case of a \Block which uses the key p.

```
\cs_new_protected:Npn \@@_Block_vi:nnnnn #1 #2 #3 #4 #5
  {
    \seq_gput_right:Ne \g_@@_blocks_seq
      {
        \l_tmpa_tl
        { \exp_not:n { #3 } }
```

Here, the curly braces for the group are mandatory.

```
        { { \exp_not:n { #4 #5 } } }
      }
  }
\cs_generate_variant:Nn \@@_Block_vi:nnnnn { e e }
```

The following macro is also for the case of a \Block which uses the key p.

```
\cs_new_protected:Npn \@@_Block_vii:nnnnn #1 #2 #3 #4 #5
  {
    \seq_gput_right:Ne \g_@@_blocks_seq
      {
        \l_tmpa_tl
        { \exp_not:n { #3 } }
        { \exp_not:n { #4 #5 } }
      }
  }
\cs_generate_variant:Nn \@@_Block_vii:nnnnn { e e }
```

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).

```
\keys_define:nn { nicematrix / Block / SecondPass }
  {
    ampersand-in-blocks .bool_set:N = \l_@@_amp_in_blocks_bool ,
    ampersand-in-blocks .default:n = true ,
    &-in-blocks .meta:n = ampersand-in-blocks ,
```

The sequence `\l_@@_tikz_seq` will contain a sequence of comma-separated lists of keys.

```
    tikz .code:n =
      \IfPackageLoadedTF { tikz }
        { \seq_put_right:Nn \l_@@_tikz_seq { { #1 } } }
        { \@@_error:n { tikz~key~without~tikz } } ,
    tikz .value_required:n = true ,
    fill .code:n =
      \tl_set_rescan:Nnn
        \l_@@_fill_tl
        { \char_set_catcode_other:N ! }
        { #1 } ,
    fill .value_required:n = true ,
```

*In fine*, the opacity will be applied by `\pgfsetfillopacity`.

```
    opacity .tl_set:N = \l_@@_opacity_tl ,
    opacity .value_required:n = true ,
    draw .code:n =
      \tl_set_rescan:Nnn
        \l_@@_draw_tl
        { \char_set_catcode_other:N ! }
        { #1 } ,
    draw .default:n = default ,
    rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
    rounded-corners .default:n = 4 pt ,
    color .code:n =
      \@@_color:n { #1 }
      \tl_set_rescan:Nnn
        \l_@@_draw_tl
        { \char_set_catcode_other:N ! }
        { #1 } ,
    borders .clist_set:N = \l_@@_borders_clist ,
    borders .value_required:n = true ,
    hvlines .meta:n = { vlines , hlines } ,
    vlines .bool_set:N = \l_@@_vlines_block_bool,
    vlines .default:n = true ,
    hlines .bool_set:N = \l_@@_hlines_block_bool,
    hlines .default:n = true ,
    line-width .dim_set:N = \l_@@_line_width_dim ,
    line-width .value_required:n = true ,
```

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

```
    j .code:n = \str_set:Nn \l_@@_hpos_block_str j
                \bool_set_true:N \l_@@_p_block_bool ,
    l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
    r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
    c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
    L .code:n = \str_set:Nn \l_@@_hpos_block_str l
                \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
    R .code:n = \str_set:Nn \l_@@_hpos_block_str r
                \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
    C .code:n = \str_set:Nn \l_@@_hpos_block_str c
                \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
    t .code:n = \str_set:Nn \l_@@_vpos_block_str t ,
    T .code:n = \str_set:Nn \l_@@_vpos_block_str T ,
    b .code:n = \str_set:Nn \l_@@_vpos_block_str b ,
    B .code:n = \str_set:Nn \l_@@_vpos_block_str B ,
    m .code:n = \str_set:Nn \l_@@_vpos_block_str c ,
    m .value_forbidden:n = true ,
    v-center .meta:n = m ,
    p .code:n = \bool_set_true:N \l_@@_p_block_bool ,
    p .value_forbidden:n = true ,
    name .tl_set:N = \l_@@_block_name_str , % .str_set:N ?
    name .value_required:n = true ,
    name .initial:n = ,
    respect-arraystretch .code:n =
      \cs_set_eq:NN \@@_reset_arraystretch: \prg_do_nothing: ,
    respect-arraystretch .value_forbidden:n = true ,
    transparent .bool_set:N = \l_@@_transparent_bool ,
    transparent .default:n = true ,
    transparent .initial:n = false ,
    unknown .code:n =
      \@@_unknown_key:nn
        { nicematrix / Block / SecondPass }
        { Unknown~key~for~Block }
  }
```

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.

```
\cs_new_protected:Npn \@@_draw_blocks:
  {
    \bool_if:NTF \c_@@_revtex_bool
      { \cs_set_eq:NN \ialign \@@_old_ialign: }
      { \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign: }
    \seq_map_inline:Nn \g_@@_blocks_seq { \@@_Block_iv:nnnnnn ##1 }
  }
```

```
\cs_new_protected:Npn \@@_Block_iv:nnnnnn #1 #2 #3 #4 #5 #6
  {
```

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

```
    \int_zero:N \l_@@_last_row_int
    \int_zero: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 follows: 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 (we write 98 for the case the the command `\Block` has been issued in the "first row").

```
    \int_compare:nNnTF { #3 } > { 98 }
      { \int_set_eq:NN \l_@@_last_row_int \c@iRow }
      { \int_set:Nn \l_@@_last_row_int { #3 } }
    \int_compare:nNnTF { #4 } > { 98 }
      { \int_set_eq:NN \l_@@_last_col_int \c@jCol }
      { \int_set:Nn \l_@@_last_col_int { #4 } }

    \int_compare:nNnTF { \l_@@_last_col_int } > { \g_@@_col_total_int }
      {
        \bool_lazy_and:nnTF
          { \l_@@_preamble_bool }
          {
            \int_compare_p:n
             { \l_@@_last_col_int <= \g_@@_static_num_of_col_int }
          }
          {
            \msg_error:nnnn { nicematrix } { Block~too~large~2 } { #1 } { #2 }
            \@@_msg_redirect_name:nn { Block~too~large~2 } { none }
            \@@_msg_redirect_name:nn { columns~not~used } { none }
          }
          { \msg_error:nnnn { nicematrix } { Block~too~large~1 } { #1 } { #2 } }
      }
      {
        \int_compare:nNnTF { \l_@@_last_row_int } > { \g_@@_row_total_int }
          { \msg_error:nnnn { nicematrix } { Block~too~large~1 } { #1 } { #2 } }
          {
            \@@_Block_v:nneenn
              { #1 }
              { #2 }
              { \int_use:N \l_@@_last_row_int }
              { \int_use:N \l_@@_last_col_int }
              { #5 }
              { #6 }
          }
      }
    }
```

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 (content) of the block.

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

The group is for the keys.

```
    \group_begin:
    \int_compare:nNnT { #1 } = { #3 }
      { \str_set:Nn \l_@@_vpos_block_str { t } }
    \keys_set:nn { nicematrix / Block / SecondPass } { #5 }
```

If the content of the block contains `&`, we will have a special treatment (since the cell must be divided in several sub-cells). Remark that `\tl_if_in:nnT` is faster then `\str_if_in:nnT`.

```
    \tl_if_in:nnT { #6 } { & } { \bool_set_true:N \l_@@_ampersand_bool }

    \bool_lazy_and:nnT
      { \l_@@_vlines_block_bool }
      { ! \l_@@_ampersand_bool }
      {
        \tl_gput_right:Ne \g_nicematrix_code_after_tl
          {
            \@@_vlines_block:nnn
              { \exp_not:n { #5 } }
              { #1 - #2 }
              { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
          }
      }
    \bool_if:NT \l_@@_hlines_block_bool
      {
        \tl_gput_right:Ne \g_nicematrix_code_after_tl
          {
            \@@_hlines_block:nnn
              { \exp_not:n { #5 } }
              { #1 - #2 }
              { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
          }
      }
    \bool_if:NF \l_@@_transparent_bool
      {
         \bool_lazy_and:nnF { \l_@@_vlines_block_bool } { \l_@@_hlines_block_bool }
          {
```

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).

```
            \seq_gput_left:Ne \g_@@_pos_of_blocks_seq
              { { #1 } { #2 } { #3 } { #4 } { \l_@@_block_name_str } }
          }
      }
```

```
    \tl_if_empty:NF \l_@@_draw_tl
      {
        \bool_lazy_or:nnT \l_@@_hlines_block_bool \l_@@_vlines_block_bool
          { \@@_error:n { hlines~with~color } }
        \tl_gput_right:Ne \g_nicematrix_code_after_tl
          {
            \@@_stroke_block:nnn
```

#5 are the options

```
              { \exp_not:n { #5 } }
              { #1 - #2 }
              { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
          }
        \seq_gput_right:Nn \g_@@_pos_of_stroken_blocks_seq
          { { #1 } { #2 } { #3 } { #4 } }
      }
    \clist_if_empty:NF \l_@@_borders_clist
      {
        \tl_gput_right:Ne \g_nicematrix_code_after_tl
          {
            \@@_stroke_borders_block:nnn
              { \exp_not:n { #5 } }
              { #1 - #2 }
              { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
          }
      }
    \tl_if_empty:NF \l_@@_fill_tl
      {
        \@@_add_opacity_to_fill:
        \tl_gput_right:Ne \g_@@_pre_code_before_tl
          {
            \@@_exp_color_arg:No \@@_roundedrectanglecolor \l_@@_fill_tl
              { #1 - #2 }
              { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
              { \dim_use:N \l_@@_rounded_corners_dim }
          }
      }
    \seq_if_empty:NF \l_@@_tikz_seq
      {
        \tl_gput_right:Ne \g_nicematrix_code_before_tl
          {
            \@@_block_tikz:nnnnn
              { \seq_use:Nn \l_@@_tikz_seq { , } }
              { #1 }
              { #2 }
              { \int_use:N \l_@@_last_row_int }
              { \int_use:N \l_@@_last_col_int }
```

We will have in that last field a list of lists of TikZ keys.

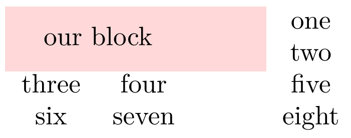
```
          }
      }

    \cs_set_protected_nopar:Npn \diagbox ##1 ##2
      {
        \tl_gput_right:Ne \g_@@_pre_code_after_tl
          {
            \@@_actually_diagbox:nnnnnn
              { #1 }
              { #2 }
              { \int_use:N \l_@@_last_row_int }
              { \int_use:N \l_@@_last_col_int }
              { \exp_not:n { ##1 } }
              { \exp_not:n { ##2 } }
            }
        }
```

Let's consider the following `{NiceTabular}`. Because of the instruction `!{\hspace{1cm}}` in the preamble which increases the space between the columns (by adding, in fact, that space to the previous column, that is to say the second column of the tabular), we will create *two* nodes relative to the block: the node `1-1-block` and the node `1-1-block-short`.

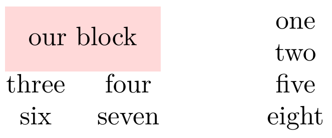
```
\begin{NiceTabular}{cc!{\hspace{1cm}}c}
\Block{2-2}{our block} &       & one    \\
                       &       & two    \\
three                  & four  & five   \\
six                    & seven & eight  \\
\end{NiceTabular}
```

We highlight the node `1-1-block`



|  |  |  |
|---|---|---|
| our block |  | one |
|  |  | two |
| three | four | five |
| six | seven | eight |

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



|  |  |  |
|---|---|---|
| our block |  | one |
|  |  | two |
| three | four | five |
| six | seven | eight |

The construction of the node corresponding to the merged cells.

```
      \pgfpicture
      \pgfrememberpicturepositiononpagetrue
      \pgf@relevantforpicturesizefalse
      \@@_qpoint:n { row - #1 }
      \dim_set_eq:NN \l_tmpa_dim \pgf@y
      \@@_qpoint:n { col - #2 }
      \dim_set_eq:NN \l_tmpb_dim \pgf@x
      \@@_qpoint:n { row - \int_eval:n { \l_@@_last_row_int + 1 } }
      \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
      \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
      \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.

```
      \@@_pgf_rect_node:nnnnn
        { \@@_env: - #1 - #2 - block }
        \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
      \str_if_empty:NF \l_@@_block_name_str
        {
          \pgfnodealias
            { \@@_env: - \l_@@_block_name_str }
            { \@@_env: - #1 - #2 - block }
          \str_if_empty:NF \l_@@_name_str
            {
              \pgfnodealias
                { \l_@@_name_str - \l_@@_block_name_str }
                { \@@_env: - #1 - #2 - block }
            }
        }
```

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.

```
      \bool_if:NF \l_@@_hpos_of_block_cap_bool
        {
          \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.

```
          \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int }
            {
```

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.

```
              \cs_if_exist:cT
                { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
                {
                  \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
                    {
                      \pgfpointanchor { \@@_env: - ##1 - #2 } { west }
                      \dim_set:Nn \l_tmpb_dim { \dim_min:nn \l_tmpb_dim \pgf@x }
                    }
                }
            }
```

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.

```
          \dim_compare:nNnT { \l_tmpb_dim } = { \c_max_dim }
            {
              \@@_qpoint:n { col - #2 }
              \dim_set_eq:NN \l_tmpb_dim \pgf@x
            }
          \dim_set:Nn \l_@@_tmpd_dim { - \c_max_dim }
          \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int }
            {
              \cs_if_exist:cT
                { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
                {
                  \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
                    {
                      \pgfpointanchor
                        { \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
                        { east }
                      \dim_set:Nn \l_@@_tmpd_dim
                        { \dim_max:nn { \l_@@_tmpd_dim } { \pgf@x } }
                    }
                }
            }
          \dim_compare:nNnT { \l_@@_tmpd_dim } = { - \c_max_dim }
            {
              \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
              \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
            }
          \@@_pgf_rect_node:nnnnn
            { \@@_env: - #1 - #2 - block - short }
            \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
        }
```

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.

```
      \bool_if:NT \l_@@_medium_nodes_bool
        {
          \@@_pgf_rect_node:nnn
            { \@@_env: - #1 - #2 - block - medium }
            { \pgfpointanchor { \@@_env: - #1 - #2 - medium } { north~west } }
            {
              \pgfpointanchor
                { \@@_env:
                  - \int_use:N \l_@@_last_row_int
                  - \int_use:N \l_@@_last_col_int - medium
                }
            { south~east }
          }
      }
    \endpgfpicture

```

`\l_@@_ampersand_bool` is raised when the content of the block actually *contents* an ampersand `&`.

```
  \bool_if:NTF \l_@@_ampersand_bool
    {
      \seq_set_split:Nnn \l_tmpa_seq { & } { #6 }
      \int_zero_new:N \l_@@_split_int
      \int_set:Nn \l_@@_split_int { \seq_count:N \l_tmpa_seq }
```

The following counters will be used to send information to `\cellcolor` if the user uses that command in a subcell.

```
      \int_zero_new:N \l_@@_first_row_int
      \int_zero_new:N \l_@@_first_col_int
      \int_zero_new:N \l_@@_last_row_int
      \int_zero_new:N \l_@@_last_col_int
      \int_set:Nn \l_@@_first_row_int { #1 }
      \int_set:Nn \l_@@_first_col_int { #2 }
      \int_set:Nn \l_@@_last_row_int { #3 }
      \int_set:Nn \l_@@_last_col_int { #4 }

      \pgfpicture
      \pgfrememberpicturepositiononpagetrue
      \pgf@relevantforpicturesizefalse
      \@@_qpoint:n { row - #1 }
      \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
      \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
      \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
      \@@_qpoint:n { col - #2 }
      \dim_set_eq:NN \l_tmpa_dim \pgf@x
      \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
      \dim_set:Nn \l_tmpb_dim
        { ( \pgf@x - \l_tmpa_dim ) / \int_use:N \l_@@_split_int }
      \bool_lazy_or:nnT
        { \l_@@_vlines_block_bool }
        { \str_if_eq_p:ee { \l_@@_vlines_clist } { all } }
        {
          \int_step_inline:nn { \l_@@_split_int - 1 }
            {
              \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
            }
        }
      \cs_set_eq:NN \cellcolor \@@_subcellcolor
      \int_zero_new:N \l_@@_split_i_int

      \str_if_eq:eeTF { \l_@@_vpos_block_str } { T }
```

```
          {
            \pgfpointanchor { \@@_env: -  #1 - #2 - block } { north }
            \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
          }
          {
            \str_if_eq:eeTF { \l_@@_vpos_block_str } { B }
              {
                \pgfpointanchor { \@@_env: -  #1 - #2 - block } { south }
                \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
              }
              {
                \bool_lazy_or:nnTF
                  { \int_compare_p:nNn { #1 } = { #3 } }
                  { \str_if_eq_p:ee { \l_@@_vpos_block_str } { t } }
                  {
                    \@@_qpoint:n { row - #1 - base }
                    \dim_set:Nn \l_@@_tmpc_dim { \pgf@y - 0.5 \arrayrulewidth }
                  }
                  {
                    \str_if_eq:eeTF { \l_@@_vpos_block_str } { b }
                      {
                        \@@_qpoint:n { row - #3 - base }
                        \dim_set:Nn \l_@@_tmpc_dim { \pgf@y - 0.5 \arrayrulewidth }
                      }
                      {
                        \@@_qpoint:n { #1 - #2 - block }
                        \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
                      }
                  }
              }
          }
        \int_step_inline:nn { \l_@@_split_int }
          {
            \group_begin:
```

The counter `\l_@@_split_i_int` is only for the command `\@@_subcellcolor`.

```
            \int_set:Nn \l_@@_split_i_int { ##1 }
            \dim_set:Nn \col@sep
              { \bool_if:NTF \l_@@_tabular_bool { \tabcolsep } { \arraycolsep } }
            \pgftransformshift
              {
                \pgfpoint
                  {
                    \l_tmpa_dim + ##1 \l_tmpb_dim -
                    \str_case:on \l_@@_hpos_block_str
                      {
                        l { \l_tmpb_dim + \col@sep}
                        c { 0.5 \l_tmpb_dim }
                        r { \col@sep }
                      }
                  }
                  { \l_@@_tmpc_dim }
              }
            \pgfset { inner~sep = \c_zero_dim }
            \pgfnode
              { rectangle }
              {
                \str_if_eq:eeTF { \l_@@_vpos_block_str } { T }
                  {
                    \str_case:on \l_@@_hpos_block_str
                      {
                        l { north~west }
                        c { north }
                        r { north~east }
                      }
                  }
                  {
                    \str_if_eq:eeTF { \l_@@_vpos_block_str } { B }
                      {
                        \str_case:on \l_@@_hpos_block_str
                          {
                            l { south~west }
                            c { south }
                            r { south~east }
                          }
                      }
                      {
                        \bool_lazy_any:nTF
                          {
                            { \int_compare_p:nNn { #1 } = { #3 } }
                            { \str_if_eq_p:ee { \l_@@_vpos_block_str } { t } }
                            { \str_if_eq_p:ee { \l_@@_vpos_block_str } { b } }
                          }
                          {
                            \str_case:on \l_@@_hpos_block_str
                              {
                                l { base~west }
                                c { base }
                                r { base~east }
                              }
                          }
                          {
                            \str_case:on \l_@@_hpos_block_str
                              {
                                l { west }
                                c { center }
                                r { east }
                              }
                          }
                      }
                  }
              }
              { \seq_item:Nn \l_tmpa_seq { ##1 } } { } { }
             \group_end:
          }
        \endpgfpicture
      }
```

Now the case where there is no ampersand & in the content of the block.

```
    {
      \bool_if:NTF \l_@@_p_block_bool
        {
```

When the final user has used the key p, we have to compute the width.

```
           \pgfpicture
             \pgfrememberpicturepositiononpagetrue
             \pgf@relevantforpicturesizefalse
             \bool_if:NTF \l_@@_hpos_of_block_cap_bool
               {
                 \@@_qpoint:n { col - #2 }
                 \dim_gset_eq:NN \g_tmpa_dim \pgf@x
                 \@@_qpoint:n { col - \int_eval:n { \l_@@_last_col_int + 1 } }
               }
               {
                 \pgfpointanchor { \@@_env: - #1 - #2 - block - short } { west }
                 \dim_gset_eq:NN \g_tmpa_dim \pgf@x
                 \pgfpointanchor { \@@_env: - #1 - #2 - block - short } { east }
               }
```

```
            \dim_gset:Nn \g_tmpb_dim { \pgf@x - \g_tmpa_dim }
          \endpgfpicture
          \hbox_set:Nn \l_@@_cell_box
            {
              \begin { minipage } [ \str_lowercase:f \l_@@_vpos_block_str ]
                { \g_tmpb_dim }
              \str_case:on \l_@@_hpos_block_str
                { c \centering r \raggedleft l \raggedright j { } }
              #6
              \end { minipage }
            }
        }
        { \hbox_set:Nn \l_@@_cell_box { \set@color #6 } }
      \bool_if:NT \g_@@_rotate_bool { \@@_rotate_cell_box: }
```

Now, we will put the label of the block. We recall that `\l_@@_vpos_block_str` is empty when the user has not used a key for the vertical position of the block.

```
      \pgfpicture
      \pgfrememberpicturepositiononpagetrue
      \pgf@relevantforpicturesizefalse
      \bool_lazy_any:nTF
        {
          { \str_if_empty_p:N \l_@@_vpos_block_str }
          { \str_if_eq_p:ee { \l_@@_vpos_block_str } { c } }
          { \str_if_eq_p:ee { \l_@@_vpos_block_str } { T } }
          { \str_if_eq_p:ee { \l_@@_vpos_block_str } { B } }
        }

        {
```

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

```
          \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`.

```
          \bool_if:nT \g_@@_last_col_found_bool
            {
              \int_compare:nNnT { #2 } = { \g_@@_col_total_int }
                { \str_set_eq:NN \l_@@_hpos_block_str \c_@@_l_str }
            }
```

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

```
          \tl_set:Ne \l_tmpa_tl
            {
              \str_case:on \l_@@_vpos_block_str
                {
```

We recall that `\l_@@_vpos_block_str` is empty when the user has not used a key for the vertical position of the block.

```
                  { } {
                        \str_case:on \l_@@_hpos_block_str
                          {
                            c { center }
                            l { west }
                            r { east }
                            j { center }
                          }
                      }
                  c {
                      \str_case:on \l_@@_hpos_block_str
                        {
                          c { center }
                          l { west }
                          r { east }
                          j { center }
                        }

                    }
                  T {
                      \str_case:on \l_@@_hpos_block_str
                        {
                          c { north }
                          l { north~west }
                          r { north~east }
                          j { north }
                        }

                    }
                  B {
                      \str_case:on \l_@@_hpos_block_str
                        {
                          c { south }
                          l { south~west }
                          r { south~east }
                          j { south }
                        }

                    }
                }
            }
          \pgftransformshift
            {
              \pgfpointanchor
                {
                  \@@_env: - #1 - #2 - block
                  \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
                }
                { \l_tmpa_tl }
            }
          \pgfset { inner~sep = \c_zero_dim }
          \pgfnode
            { rectangle }
            { \l_tmpa_tl }
            { \box_use_drop:N \l_@@_cell_box } { } { }
        }
```

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

```
        {
          \pgfextracty \l_tmpa_dim
            {
              \@@_qpoint:n
                {
                  row - \str_if_eq:eeTF { \l_@@_vpos_block_str } { b } { #3 } { #1 }
                  - base
                }
            }
          \dim_sub:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }
```

We retrieve (in `\pgf@x`) the $x$-value of the center of the block.

```
          \pgfpointanchor
            {
              \@@_env: - #1 - #2 - block
              \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
            }
            {
              \str_case:on \l_@@_hpos_block_str
                {
                  c { center }
```

```
                    l { west }
                    r { east }
                    j { center }
                  }
              }
```

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

```
            \pgftransformshift { \pgfpoint \pgf@x \l_tmpa_dim }
            \pgfset { inner~sep = \c_zero_dim }
            \pgfnode
              { rectangle }
              {
                \str_case:on \l_@@_hpos_block_str
                  {
                    c { base }
                    l { base~west }
                    r { base~east }
                    j { base }
                  }
              }
              { \box_use_drop:N \l_@@_cell_box } { } { }
          }
          \endpgfpicture
        }
      \group_end:
    }
\cs_generate_variant:Nn \@@_Block_v:nnnnnn { n n e e }
```

The following command adds the value of `\l_@@_opacity_tl` (if not empty) to the specification of color set in `\l_@@_fill_tl` (the information of opacity is added in between square brackets before the color itself).

```
\cs_new_protected:Npn \@@_add_opacity_to_fill:
  {
    \tl_if_empty:NF \l_@@_opacity_tl
      {
        \tl_if_head_eq_meaning:oNTF \l_@@_fill_tl [
          {
            \tl_set:Ne \l_@@_fill_tl
              {
                [ opacity = \l_@@_opacity_tl ,
                \tl_tail:o \l_@@_fill_tl
              }
          }
          {
            \tl_set:Ne \l_@@_fill_tl
              { [ opacity = \l_@@_opacity_tl ] { \exp_not:o \l_@@_fill_tl } }
          }
      }
  }
```

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).

```
\cs_new_protected:Npn \@@_stroke_block:nnn #1 #2 #3
  {
    \group_begin:
    \tl_clear:N \l_@@_draw_tl
    \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
    \keys_set_known:nn { nicematrix / BlockStroke } { #1 }
    \pgfpicture
    \pgfrememberpicturepositiononpagetrue
    \pgf@relevantforpicturesizefalse
    \tl_if_empty:NF \l_@@_draw_tl
      {
```

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

```
        \tl_if_eq:NnTF \l_@@_draw_tl { default }
          { \CT@arc@ }
          { \@@_color:o \l_@@_draw_tl }
      }
    \pgfsetcornersarced
      {
        \pgfpoint
          { \l_@@_rounded_corners_dim }
          { \l_@@_rounded_corners_dim }
      }
    \@@_cut_on_hyphen:w #2 \q_stop
    \int_compare:nNnF { \l_tmpa_tl } > { \c@iRow }
      {
        \int_compare:nNnF { \l_tmpb_tl } > { \c@jCol }
          {
            \@@_qpoint:n { row - \l_tmpa_tl }
            \dim_set_eq:NN \l_tmpb_dim \pgf@y
            \@@_qpoint:n { col - \l_tmpb_tl }
            \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
            \@@_cut_on_hyphen:w #3 \q_stop
            \int_compare:nNnT { \l_tmpa_tl } > { \c@iRow }
              { \tl_set:No \l_tmpa_tl { \int_use:N \c@iRow } }
            \int_compare:nNnT { \l_tmpb_tl } > { \c@jCol }
              { \tl_set:No \l_tmpb_tl { \int_use:N \c@jCol } }
            \@@_qpoint:n { row - \int_eval:n { \l_tmpa_tl + 1 } }
            \dim_set_eq:NN \l_tmpa_dim \pgf@y
            \@@_qpoint:n { col - \int_eval:n { \l_tmpb_tl + 1 } }
            \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
            \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
            \pgfpathrectanglecorners
              { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
              { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
            \dim_compare:nNnTF { \l_@@_rounded_corners_dim } = { \c_zero_dim }
              { \pgfusepathqstroke }
              { \pgfusepath { stroke } }
          }
      }
    \endpgfpicture
    \group_end:
  }
```

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

```
\keys_define:nn { nicematrix / BlockStroke }
  {
    color .tl_set:N = \l_@@_draw_tl ,
    draw .code:n =
      \tl_if_empty:eF { #1 } { \tl_set:Nn \l_@@_draw_tl { #1 } } ,
    draw .default:n = default ,
    line-width .dim_set:N = \l_@@_line_width_dim ,
    rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
    rounded-corners .default:n = 4 pt
  }
```

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).

```
\cs_new_protected:Npn \@@_vlines_block:nnn #1 #2 #3
  {
    \group_begin:
    \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
    \keys_set_known:nn { nicematrix / BlockBorders } { #1 }
    \dim_set_eq:NN \arrayrulewidth \l_@@_line_width_dim
    \@@_cut_on_hyphen:w #2 \q_stop
    \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
    \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
    \@@_cut_on_hyphen:w #3 \q_stop
    \tl_set:Ne \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
    \tl_set:Ne \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
    \int_step_inline:nnn { \l_@@_tmpd_tl } { \l_tmpb_tl }
      {
        \use:e
          {
            \@@_vline:n
              {
                position = ##1 ,
                start = \l_@@_tmpc_tl ,
                end = \int_eval:n { \l_tmpa_tl - 1 } ,
                total-width = \dim_use:N \l_@@_line_width_dim
              }
          }
      }
    \group_end:
  }
\cs_new_protected:Npn \@@_hlines_block:nnn #1 #2 #3
  {
    \group_begin:
    \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
    \keys_set_known:nn { nicematrix / BlockBorders } { #1 }
    \dim_set_eq:NN \arrayrulewidth \l_@@_line_width_dim
    \@@_cut_on_hyphen:w #2 \q_stop
    \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
    \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
    \@@_cut_on_hyphen:w #3 \q_stop
    \tl_set:Ne \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
    \tl_set:Ne \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
    \int_step_inline:nnn { \l_@@_tmpc_tl } { \l_tmpa_tl }
      {
        \use:e
          {
            \@@_hline:n
              {
                position = ##1 ,
                start = \l_@@_tmpd_tl ,
                end = \int_eval:n { \l_tmpb_tl - 1 } ,
                total-width = \dim_use:N \l_@@_line_width_dim
              }
          }
      }
    \group_end:
  }
```

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).

```
\cs_new_protected:Npn \@@_stroke_borders_block:nnn #1 #2 #3
  {
    \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
    \keys_set_known:nn { nicematrix / BlockBorders } { #1 }
    \dim_compare:nNnTF { \l_@@_rounded_corners_dim } > { \c_zero_dim }
      { \@@_error:n { borders~forbidden } }
      {
        \tl_clear_new:N \l_@@_borders_tikz_tl
        \keys_set:no
          { nicematrix / OnlyForTikzInBorders }
          \l_@@_borders_clist
        \@@_cut_on_hyphen:w #2 \q_stop
        \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
        \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
        \@@_cut_on_hyphen:w #3 \q_stop
        \tl_set:Ne \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
        \tl_set:Ne \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
        \@@_stroke_borders_block_i:
      }
  }
\hook_gput_code:nnn { begindocument } { . }
  {
    \cs_new_protected:Npe \@@_stroke_borders_block_i:
      {
        \c_@@_pgfortikzpicture_tl
        \@@_stroke_borders_block_ii:
        \c_@@_endpgfortikzpicture_tl
      }
  }
\cs_new_protected:Npn \@@_stroke_borders_block_ii:
  {
    \pgfrememberpicturepositiononpagetrue
    \pgf@relevantforpicturesizefalse
    \CT@arc@
    \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
    \clist_if_in:NnT \l_@@_borders_clist { right }
      { \@@_stroke_vertical:n \l_tmpb_tl }
    \clist_if_in:NnT \l_@@_borders_clist { left }
      { \@@_stroke_vertical:n \l_@@_tmpd_tl }
    \clist_if_in:NnT \l_@@_borders_clist { bottom }
      { \@@_stroke_horizontal:n \l_tmpa_tl }
    \clist_if_in:NnT \l_@@_borders_clist { top }
      { \@@_stroke_horizontal:n \l_@@_tmpc_tl }
  }
\keys_define:nn { nicematrix / OnlyForTikzInBorders }
  {
    tikz .code:n =
      \cs_if_exist:NTF \tikzpicture
        { \tl_set:Nn \l_@@_borders_tikz_tl { #1 } }
        { \@@_error:n { tikz~in~borders~without~tikz } } ,
    tikz .value_required:n = true ,
    top .code:n = ,
    bottom .code:n = ,
    left .code:n = ,
    right .code:n = ,
    unknown .code:n = \@@_error:n { bad~border }
  }
```

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).

```
\cs_new_protected:Npn \@@_stroke_vertical:n #1
  {
    \@@_qpoint:n \l_@@_tmpc_tl
    \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
    \@@_qpoint:n \l_tmpa_tl
    \dim_set:Nn \l_@@_tmpc_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
    \@@_qpoint:n { #1 }
    \tl_if_empty:NTF \l_@@_borders_tikz_tl
```

```
      {
        \pgfpathmoveto { \pgfpoint \pgf@x \l_tmpb_dim }
        \pgfpathlineto { \pgfpoint \pgf@x \l_@@_tmpc_dim }
        \pgfusepathqstroke
      }
      {
        \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
          ( \pgf@x , \l_tmpb_dim ) -- ( \pgf@x , \l_@@_tmpc_dim ) ;
      }
  }
```

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).

```
\cs_new_protected:Npn \@@_stroke_horizontal:n #1
  {
    \@@_qpoint:n \l_@@_tmpd_tl
    \clist_if_in:NnTF \l_@@_borders_clist { left }
      { \dim_set:Nn \l_tmpa_dim { \pgf@x - 0.5 \l_@@_line_width_dim } }
      { \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \l_@@_line_width_dim } }
    \@@_qpoint:n \l_tmpb_tl
    \dim_set:Nn \l_tmpb_dim { \pgf@x + 0.5 \l_@@_line_width_dim }
    \@@_qpoint:n { #1 }
    \tl_if_empty:NTF \l_@@_borders_tikz_tl
      {
        \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }
        \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
        \pgfusepathqstroke
      }
      {
        \use:e { \exp_not:N \draw [ \l_@@_borders_tikz_tl ] }
          ( \l_tmpa_dim , \pgf@y ) -- ( \l_tmpb_dim , \pgf@y ) ;
      }
  }
```

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

```
\keys_define:nn { nicematrix / BlockBorders }
  {
    borders .clist_set:N = \l_@@_borders_clist ,
    rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
    rounded-corners .default:n = 4 pt ,
    line-width .dim_set:N = \l_@@_line_width_dim
  }
```

The following command will be used if the key `tikz` has been used for the command `\Block`.
`#1` is a *list of lists* of TikZ keys used with the path.
*Example*: `{{offset=1pt,draw,red},{offset=2pt,draw,blue}}`
which arises from a command such as :
`\Block[tikz={offset=1pt,draw,red},tikz={offset=2pt,draw,blue}]{2-2}{}`
The arguments `#2` and `#3` are the coordinates of the first cell and `#4` and `#5` the coordinates of the last cell of the block.

```
\cs_new_protected:Npn \@@_block_tikz:nnnnn #1 #2 #3 #4 #5
  {
    \begin { tikzpicture }
    \@@_clip_with_rounded_corners:
```

We use `clist_map_inline:nn` because `#5` is a list of lists.

```
    \clist_map_inline:nn { #1 }
      {
```

We extract the key `offset` which is *not* a key of TikZ but a key added by nicematrix.

```
        \keys_set_known:nnN { nicematrix / SpecialOffset } { ##1 } \l_tmpa_tl
        \use:e { \exp_not:N \path [ \l_tmpa_tl ] }
              (
                [
                  xshift = \dim_use:N \l_@@_offset_dim ,
                  yshift = - \dim_use:N \l_@@_offset_dim
                ]
                #2 -| #3
              )
              rectangle
              (
                [
                  xshift = - \dim_use:N \l_@@_offset_dim ,
                  yshift = \dim_use:N \l_@@_offset_dim
                ]
                \int_eval:n { #4 + 1 } -| \int_eval:n { #5 + 1 }
              ) ;
      }
    \end { tikzpicture }
  }
\cs_generate_variant:Nn \@@_block_tikz:nnnnn { o }

\keys_define:nn { nicematrix / SpecialOffset }
  { offset .dim_set:N = \l_@@_offset_dim }
```

In some circonstancies, we want to nullify the command `\Block`. In order to reach that goal, we will link the command `\Block` to the following command `\@@_NullBlock:` which has the same syntax as the standard command `\Block` but which is no-op.

```
\cs_new_protected:Npn \@@_NullBlock:
  { \@@_collect_options:n { \@@_NullBlock_i: } }
\NewExpandableDocumentCommand \@@_NullBlock_i: { m m D < > { } +m }
  { }
```

The following command will be linked to `\cellcolor` in the sub-cells of a block which contains ampersands (`&`). Of course, `&-in-blocks` must be in force.

```
\NewDocumentCommand \@@_subcellcolor { O { } m }
  {
    \tl_gput_right:Ne \g_@@_pre_code_before_tl
      {
```

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).

```
        \@@_subcellcolor:nnnnnnn
          {
            \tl_if_blank:nTF { #1 }
              { { \exp_not:n { #2 } } }
              { [ #1 ] { \exp_not:n { #2 } } }
          }
          { \int_use:N \l_@@_first_row_int } % first row of the block
          { \int_use:N \l_@@_first_col_int } % first column of the block
          { \int_use:N \l_@@_last_row_int }  % last row of the block
          { \int_use:N \l_@@_last_col_int }  % last column of the block
          { \int_use:N \l_@@_split_int }
          { \int_use:N \l_@@_split_i_int }
      }
    \ignorespaces
  }
\cs_new_protected:Npn \@@_subcellcolor:nnnnnnn #1 #2 #3 #4 #5 #6 #7
  {
    \@@_color_opacity: #1
    \pgfpicture
    \pgf@relevantforpicturesizefalse
    \@@_qpoint:n { col - #3 }
    \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
    \@@_qpoint:n { col - \int_eval:n { #5 + 1 } }
    \dim_set:Nn \l_tmpa_dim { ( \pgf@x - \l_@@_tmpc_dim ) / #6 }
    \dim_set:Nn \l_tmpb_dim { \l_@@_tmpc_dim + #7 \l_tmpa_dim }
    \@@_qpoint:n { row - \int_eval:n { #4 + 1 } }
    \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
    \@@_qpoint:n { row - #2 }
    \pgfpathrectanglecorners
      { \pgfpoint { \l_tmpb_dim - \l_tmpa_dim } { \l_@@_tmpc_dim } }
      { \pgfpoint { \l_tmpb_dim } { \pgf@y } }
    \pgfusepathqfill
    \endpgfpicture
  }
```

## 27 How to draw the dotted lines transparently

```
\cs_set_protected:Npn \@@_renew_matrix:
  {
    \RenewDocumentEnvironment { pmatrix } { }
      { \pNiceMatrix }
      { \endpNiceMatrix }
    \RenewDocumentEnvironment { vmatrix } { }
      { \vNiceMatrix }
      { \endvNiceMatrix }
    \RenewDocumentEnvironment { Vmatrix } { }
      { \VNiceMatrix }
      { \endVNiceMatrix }
    \RenewDocumentEnvironment { bmatrix } { }
      { \bNiceMatrix }
      { \endbNiceMatrix }
    \RenewDocumentEnvironment { Bmatrix } { }
      { \BNiceMatrix }
      { \endBNiceMatrix }
  }
```

## 28 Automatic arrays

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

```
\keys_define:nn { nicematrix / Auto }
  {
    columns-type .tl_set:N = \l_@@_columns_type_tl ,
    columns-type .value_required:n = true ,
    l .meta:n = { columns-type = l } ,
    r .meta:n = { columns-type = r } ,
    c .meta:n = { columns-type = c } ,
    delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
    delimiters / color .value_required:n = true ,
    delimiters / max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
    delimiters / max-width .default:n = true ,
    delimiters .code:n = \keys_set:nn { nicematrix / delimiters } { #1 } ,
    delimiters .value_required:n = true ,
    rounded-corners .dim_set:N = \l_@@_tab_rounded_corners_dim ,
    rounded-corners .default:n = 4 pt
  }
```

```
\NewDocumentCommand \AutoNiceMatrixWithDelims
  { m m O { } > { \SplitArgument { 1 } { - } } m O { } m ! O { } }
  { \@@_auto_nice_matrix:nnnnnn { #1 } { #2 } #4 { #6 } { #3 , #5 , #7  } }
\cs_new_protected:Npn \@@_auto_nice_matrix:nnnnnn #1 #2 #3 #4 #5 #6
  {
```

The group is for the protection of the keys.

```
    \group_begin:
    \keys_set_known:nnN { nicematrix / Auto } { #6 } \l_tmpa_tl
    \use:e
      {
        \exp_not:N \begin { NiceArrayWithDelims } { #1 } { #2 }
          { * { #4 } { \exp_not:o \l_@@_columns_type_tl } }
          [ \exp_not:o \l_tmpa_tl ]
      }
    \int_if_zero:nT { \l_@@_first_row_int }
      {
        \int_if_zero:nT { \l_@@_first_col_int } { & }
        \prg_replicate:nn { #4 - 1 } { & }
        \int_compare:nNnT { \l_@@_last_col_int } > { -1 } { & } \\
      }
    \prg_replicate:nn { #3 }
      {
        \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`).

```
        \prg_replicate:nn { #4 - 1 } { { } #5 & } #5
        \int_compare:nNnT { \l_@@_last_col_int } > { -1 } { & } \\
      }
    \int_compare:nNnT { \l_@@_last_row_int } > { -2 }
      {
        \int_if_zero:nT { \l_@@_first_col_int } { & }
        \prg_replicate:nn { #4 - 1 } { & }
        \int_compare:nNnT { \l_@@_last_col_int } > { -1 } { & } \\
      }
    \end { NiceArrayWithDelims }
    \group_end:
  }
\cs_set_protected:Npn \@@_define_com:NNN #1 #2 #3
  {
    \cs_set_protected:cpn { #1 AutoNiceMatrix }
      {
        \bool_gset_true:N \g_@@_delims_bool
        \str_gset:Ne \g_@@_name_env_str { #1 AutoNiceMatrix }
        \AutoNiceMatrixWithDelims { #2 } { #3 }
      }
  }
```

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

```
\NewDocumentCommand \AutoNiceMatrix { O { } m O { } m ! O { } }
  {
    \group_begin:
    \bool_gset_false:N \g_@@_delims_bool
    \AutoNiceMatrixWithDelims . . { #2 } { #4 } [ #1 , #3 , #5 ]
    \group_end:
  }
```

## 29 The redefinition of the command \dotfill

```
\cs_set_eq:NN \@@_old_dotfill: \dotfill
\cs_new_protected:Npn \@@_dotfill:
  {
```

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}`).

```
    \@@_old_dotfill:
    \tl_gput_right:Nn \g_@@_cell_after_hook_tl \@@_dotfill_i:
  }
```

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`.

```
\cs_new_protected:Npn \@@_dotfill_i:
  {
    \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } = { \c_zero_dim }
      { \@@_old_dotfill: }
  }
```

## 30 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.

```
\cs_new_protected:Npn \@@_diagbox:nn #1 #2
  {
    \tl_gput_right:Ne \g_@@_pre_code_after_tl
      {
        \@@_actually_diagbox:nnnnnn
          { \int_use:N \c@iRow }
          { \int_use:N \c@jCol }
          { \int_use:N \c@iRow }
          { \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 chunk of instructions.

```
          { \g_@@_row_style_tl \exp_not:n { #1 } }
          { \g_@@_row_style_tl \exp_not:n { #2 } }
      }
```

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`.

```
    \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
      {
        { \int_use:N \c@iRow }
        { \int_use:N \c@jCol }
        { \int_use:N \c@iRow }
        { \int_use:N \c@jCol }
```

The last argument is for the name of the block.

```
        { }
      }
  }
```

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.

```
\cs_new_protected:Npn \@@_actually_diagbox:nnnnnn #1 #2 #3 #4 #5 #6
  {
    \pgfpicture
    \pgf@relevantforpicturesizefalse
    \pgfrememberpicturepositiononpagetrue
    \@@_qpoint:n { row - #1 }
    \dim_set_eq:NN \l_tmpa_dim \pgf@y
    \@@_qpoint:n { col - #2 }
    \dim_set_eq:NN \l_tmpb_dim \pgf@x
    \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
    \@@_qpoint:n { row - \int_eval:n { #3 + 1 } }
    \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
    \@@_qpoint:n { col - \int_eval:n { #4 + 1 } }
    \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
    \pgfpathlineto { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
    {
```

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.

```
      \CT@arc@
      \pgfsetroundcap
      \pgfusepathqstroke
    }
    \pgfset { inner~sep = 1 pt }
    \pgfscope
    \pgftransformshift { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
    \pgfnode { rectangle } { south~west }
      {
        \begin { minipage } { 20 cm }
```

The `\scan_stop:` avoids an error in math mode when the argument `#5` is empty.

```
        \@@_math_toggle: \scan_stop: #5 \@@_math_toggle:
        \end { minipage }
      }
      { }
      { }
    \endpgfscope
    \pgftransformshift { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
    \pgfnode { rectangle } { north~east }
      {
        \begin { minipage } { 20 cm }
        \raggedleft
        \@@_math_toggle: \scan_stop: #6 \@@_math_toggle:
        \end { minipage }
      }
      { }
      { }
    \endpgfpicture
  }
```

## 31 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. 90.

In the environments of nicematrix, `\CodeAfter` will be linked to `\@@_CodeAfter:`. That macro must *not* be protected since it begins with `\omit`.

```
\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 `\\`.

```
\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`.

```
\cs_new_protected:Npn \@@_CodeAfter_ii:n #1 \end
  {
    \tl_gput_right:Nn \g_nicematrix_code_after_tl { #1 }
    \@@_CodeAfter_iv:n
  }
```

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

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

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.

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

```
      {
        \tl_gput_right:Nn \g_nicematrix_code_after_tl { \end { #1 } }
        \@@_CodeAfter_ii:n
      }
  }
```

## 32 The delimiters in the preamble

The command `\@@_delimiter:nnn` will be used to draw delimiters inside the matrix when delimiters are specified in the preamble of the array. It does *not* concern the exterior delimiters added by `{NiceArrayWithDelims}` (and `{pNiceArray}`, `{pNiceMatrix}`, etc.).

A delimiter in the preamble of the array will write an instruction `\@@_delimiter:nnn` in the `\g_@@_pre_code_after_tl` (and also potentially add instructions in the preamble provided to `\array` in order to add space between columns).

The first argument is the type of delimiter (`(`, `[`, `\{`, `)`, `]` or `\}`). The second argument is the number of column. The third argument is a boolean equal to `\c_true_bool` (resp. `\c_false_true`) when the delimiter must be put on the left (resp. right) side.

```
\cs_new_protected:Npn \@@_delimiter:nnn #1 #2 #3
  {
    \pgfpicture
    \pgfrememberpicturepositiononpagetrue
    \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.

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

```
    \bool_if:nTF { #3 }
      { \dim_set_eq:NN \l_tmpa_dim \c_max_dim }
      { \dim_set:Nn \l_tmpa_dim { - \c_max_dim } }
    \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int }
      {
        \cs_if_exist:cT
          { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
          {
            \pgfpointanchor
              { \@@_env: - ##1 - #2 }
              { \bool_if:nTF { #3 } { west } { east } }
            \dim_set:Nn \l_tmpa_dim
              {
                \bool_if:nTF { #3 }
                  { \dim_min:nn }
                  { \dim_max:nn }
                \l_tmpa_dim
                { \pgf@x }
              }
          }
      }
```

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

```
    \pgfset { inner~sep = \c_zero_dim }
    \dim_zero:N \nulldelimiterspace
    \pgftransformshift
      {
        \pgfpoint
          { \l_tmpa_dim }
          { ( \l_@@_y_initial_dim + \l_@@_y_final_dim + \arrayrulewidth ) / 2 }
      }
    \pgfnode
      { rectangle }
      { \bool_if:nTF { #3 } { east } { west } }
      {
```

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

```
        \nullfont
        $ % $
        \@@_color:o \l_@@_delimiters_color_tl
        \bool_if:nTF { #3 } { \left #1 } { \left . }
        \vcenter
          {
            \nullfont
            \hrule \@height
                   \dim_eval:n { \l_@@_y_initial_dim - \l_@@_y_final_dim }
                   \@depth \c_zero_dim
                   \@width \c_zero_dim
          }
        \bool_if:nTF { #3 } { \right . } { \right #1 }
        $ % $
      }
      { }
      { }
    \endpgfpicture
  }
```

## 33 The command \SubMatrix

```
\keys_define:nn { nicematrix / sub-matrix }
  {
    extra-height .dim_set:N = \l_@@_submatrix_extra_height_dim ,
    extra-height .value_required:n = true ,
    left-xshift .dim_set:N = \l_@@_submatrix_left_xshift_dim ,
    left-xshift .value_required:n = true ,
    right-xshift .dim_set:N = \l_@@_submatrix_right_xshift_dim ,
    right-xshift .value_required:n = true ,
    xshift .meta:n = { left-xshift = #1, right-xshift = #1 } ,
    xshift .value_required:n = true ,
    delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
    delimiters / color .value_required:n = true ,
    slim .bool_set:N = \l_@@_submatrix_slim_bool ,
    slim .default:n = true ,
    hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
    hlines .default:n = all ,
    vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
    vlines .default:n = all ,
    hvlines .meta:n = { hlines, vlines } ,
    hvlines .value_forbidden:n = true
  }
\keys_define:nn { nicematrix }
  {
    SubMatrix .inherit:n = nicematrix / sub-matrix ,
    NiceArray / sub-matrix .inherit:n = nicematrix / sub-matrix ,
    pNiceArray / sub-matrix .inherit:n = nicematrix / sub-matrix ,
    NiceMatrixOptions / sub-matrix .inherit:n = nicematrix / sub-matrix ,
  }
```

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

```
\keys_define:nn { nicematrix / SubMatrix }
  {
    delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
    delimiters / color .value_required:n = true ,
    hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
    hlines .default:n = all ,
    vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
    vlines .default:n = all ,
    hvlines .meta:n = { hlines, vlines } ,
    hvlines .value_forbidden:n = true ,
    name .code:n =
      \tl_if_empty:nTF { #1 }
        { \@@_error:n { Invalid~name } }
        {
          \regex_if_match:nnTF { \A[A-Za-z][A-Za-z0-9]*\Z } { #1 }
            {
              \seq_if_in:NnTF \g_@@_submatrix_names_seq { #1 }
                { \@@_error:nn { Duplicate~name~for~SubMatrix } { #1 } }
                {
                  \str_set:Nn \l_@@_submatrix_name_str { #1 }
                  \seq_gput_right:Nn \g_@@_submatrix_names_seq { #1 }
                }
            }
            { \@@_error:n { Invalid~name } }
        } ,
    name .value_required:n = true ,
    rules .code:n = \keys_set:nn { nicematrix / rules } { #1 } ,
    rules .value_required:n = true ,
    code .tl_set:N = \l_@@_code_tl ,
    code .value_required:n = true ,
    unknown .code:n = \@@_error:n { Unknown~key~for~SubMatrix }
  }
```

```
\NewDocumentCommand \@@_SubMatrix_in_code_before { m m m m ! O { } }
  {
    \tl_gput_right:Ne \g_@@_pre_code_after_tl
      {
        \SubMatrix { #1 } { #2 } { #3 } { #4 }
          [
            delimiters / color = \l_@@_delimiters_color_tl ,
            hlines = \l_@@_submatrix_hlines_clist ,
            vlines = \l_@@_submatrix_vlines_clist ,
            extra-height = \dim_use:N \l_@@_submatrix_extra_height_dim ,
            left-xshift = \dim_use:N \l_@@_submatrix_left_xshift_dim ,
            right-xshift = \dim_use:N \l_@@_submatrix_right_xshift_dim ,
            slim = \bool_to_str:N \l_@@_submatrix_slim_bool ,
            #5
          ]
      }
    \@@_SubMatrix_in_code_before_i { #2 } { #3 }
    \ignorespaces
  }
```

```
\NewDocumentCommand \@@_SubMatrix_in_code_before_i
  { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
  { \@@_SubMatrix_in_code_before_i:nnnn #1 #2 }
```

```
\cs_new_protected:Npn \@@_SubMatrix_in_code_before_i:nnnn #1 #2 #3 #4
  {
    \seq_gput_right:Ne \g_@@_submatrix_seq
      {
```

We use `\str_if_eq:eeTF` because it is fully expandable (and slightly faster than `\tl_if_eq:nnTF`).

```
        { \str_if_eq:eeTF { #1 } { last } { \int_use:N \c@iRow } { #1 } }
        { \str_if_eq:eeTF { #2 } { last } { \int_use:N \c@jCol } { #2 } }
        { \str_if_eq:eeTF { #3 } { last } { \int_use:N \c@iRow } { #3 } }
        { \str_if_eq:eeTF { #4 } { last } { \int_use:N \c@jCol } { #4 } }
      }
  }
```

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`).

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

```
\cs_new_protected:Npn \@@_compute_i_j:nnnn #1 #2 #3 #4
  {
    \def \l_@@_first_i_tl { #1 }
    \def \l_@@_first_j_tl { #2 }
    \def \l_@@_last_i_tl { #3 }
    \def \l_@@_last_j_tl { #4 }
    \tl_if_eq:NnT \l_@@_first_i_tl { last }
      { \tl_set:NV \l_@@_first_i_tl \c@iRow }
    \tl_if_eq:NnT \l_@@_first_j_tl { last }
      { \tl_set:NV \l_@@_first_j_tl \c@jCol }
    \tl_if_eq:NnT \l_@@_last_i_tl { last }
      { \tl_set:NV \l_@@_last_i_tl \c@iRow }
    \tl_if_eq:NnT \l_@@_last_j_tl { last }
      { \tl_set:NV \l_@@_last_j_tl \c@jCol }
  }
```

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`.

```
\hook_gput_code:nnn { begindocument } { . }
  {
    \tl_set_rescan:Nnn \l_tmpa_tl { } { m m m m O { } E { _ ^ } { { } { } } }
    \exp_args:NNo \NewDocumentCommand \@@_SubMatrix \l_tmpa_tl
      { \@@_sub_matrix:nnnnnnn { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } { #7 } }
  }
\cs_new_protected:Npn \@@_sub_matrix:nnnnnnn #1 #2 #3 #4 #5 #6 #7
  {
    \group_begin:
```

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

```
    \@@_compute_i_j:nn { #2 } { #3 }
    \int_compare:nNnT { \l_@@_first_i_tl } = { \l_@@_last_i_tl }
      { \def \arraystretch { 1 } }
    \bool_lazy_or:nnTF
      { \int_compare_p:nNn { \l_@@_last_i_tl } > { \g_@@_row_total_int } }
      { \int_compare_p:nNn { \l_@@_last_j_tl } > { \g_@@_col_total_int } }
      { \@@_error:nn { Construct~too~large } { \SubMatrix } }
      {
        \str_clear_new:N \l_@@_submatrix_name_str
        \keys_set:nn { nicematrix / SubMatrix } { #5 }
        \pgfpicture
        \pgfrememberpicturepositiononpagetrue
        \pgf@relevantforpicturesizefalse
        \pgfset { inner~sep = \c_zero_dim }
        \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
        \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
```

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

```
        \bool_if:NTF \l_@@_submatrix_slim_bool
          { \int_step_inline:nnn { \l_@@_first_i_tl } { \l_@@_last_i_tl } }
          { \int_step_inline:nnn { \l_@@_first_row_int } { \g_@@_row_total_int } }
          {
            \cs_if_exist:cT
              { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
              {
                \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
                \dim_compare:nNnT { \pgf@x } < { \l_@@_x_initial_dim }
                  { \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x }
              }
            \cs_if_exist:cT
              { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
              {
                \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
                \dim_compare:nNnT { \pgf@x } > { \l_@@_x_final_dim }
                  { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
              }
          }
        \dim_compare:nNnTF { \l_@@_x_initial_dim } = { \c_max_dim }
          { \@@_error:nn { Impossible~delimiter } { left } }
          {
            \dim_compare:nNnTF { \l_@@_x_final_dim } = { - \c_max_dim }
```

```
              { \@@_error:nn { Impossible~delimiter } { right } }
              { \@@_sub_matrix_i:nnnn { #1 } { #4 } { #6 } { #7 } }
          }
        \endpgfpicture
      }
    \group_end:
    \ignorespaces
  }
```

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

```
\cs_new_protected:Npn \@@_sub_matrix_i:nnnn #1 #2 #3 #4
  {
    \@@_qpoint:n { row - \l_@@_first_i_tl - base }
    \dim_set:Nn \l_@@_y_initial_dim
      {
        \fp_to_dim:n
          {
            \pgf@y
            + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch
          }
      }
    \@@_qpoint:n { row - \l_@@_last_i_tl - base }
    \dim_set:Nn \l_@@_y_final_dim
      { \fp_to_dim:n { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch } }
    \int_step_inline:nnn { \l_@@_first_col_int } { \g_@@_col_total_int }
      {
        \cs_if_exist:cT
          { pgf @ sh @ ns @ \@@_env: - \l_@@_first_i_tl - ##1 }
          {
            \pgfpointanchor { \@@_env: - \l_@@_first_i_tl - ##1 } { north }
            \dim_set:Nn \l_@@_y_initial_dim
              { \dim_max:nn { \l_@@_y_initial_dim } { \pgf@y } }
          }
        \cs_if_exist:cT
          { pgf @ sh @ ns @ \@@_env: - \l_@@_last_i_tl - ##1 }
          {
            \pgfpointanchor { \@@_env: - \l_@@_last_i_tl - ##1 } { south }
            \dim_compare:nNnT { \pgf@y } < { \l_@@_y_final_dim }
              { \dim_set_eq:NN \l_@@_y_final_dim \pgf@y }
          }
      }
    \dim_set:Nn \l_tmpa_dim
      {
        \l_@@_y_initial_dim - \l_@@_y_final_dim +
        \l_@@_submatrix_extra_height_dim - \arrayrulewidth
      }
    \dim_zero:N \nulldelimiterspace
```

We will draw the rules in the \SubMatrix.

```
    \group_begin:
    \pgfsetlinewidth { 1.1 \arrayrulewidth }
    \@@_set_CTarc:o \l_@@_rules_color_tl
    \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.

```
    \seq_map_inline:Nn \g_@@_cols_vlism_seq
      {
        \int_compare:nNnT { \l_@@_first_j_tl } < { ##1 }
          {
            \int_compare:nNnT
              { ##1 } < { \int_eval:n { \l_@@_last_j_tl + 1 } }
```

```
                {
```

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

```
                  \@@_qpoint:n { col - ##1 }
                  \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
                  \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
                  \pgfusepathqstroke
                }
            }
        }
```

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.

```
      \str_if_eq:eeTF { \l_@@_submatrix_vlines_clist } { all }
        { \int_step_inline:nn { \l_@@_last_j_tl - \l_@@_first_j_tl } }
        { \clist_map_inline:Nn \l_@@_submatrix_vlines_clist }
        {
          \bool_lazy_and:nnTF
            { \int_compare_p:nNn { ##1 } > { \c_zero_int } }
            {
               \int_compare_p:nNn
                 { ##1 } < { \l_@@_last_j_tl - \l_@@_first_j_tl + 1 } }
            {
              \@@_qpoint:n { col - \int_eval:n { ##1 + \l_@@_first_j_tl } }
              \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
              \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
              \pgfusepathqstroke
            }
            { \@@_error:nnn { Wrong~line~in~SubMatrix } { vertical } { ##1 } }
        }
```

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.

```
      \str_if_eq:eeTF { \l_@@_submatrix_hlines_clist } { all }
        { \int_step_inline:nn { \l_@@_last_i_tl - \l_@@_first_i_tl } }
        { \clist_map_inline:Nn \l_@@_submatrix_hlines_clist }
        {
          \bool_lazy_and:nnTF
            { \int_compare_p:nNn { ##1 } > { \c_zero_int } }
            {
              \int_compare_p:nNn
                { ##1 } < { \l_@@_last_i_tl - \l_@@_first_i_tl + 1 } }
            {
              \@@_qpoint:n { row - \int_eval:n { ##1 + \l_@@_first_i_tl } }
```

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

```
              \group_begin:
```

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

```
              \dim_set:Nn \l_tmpa_dim
                { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
              \str_case:nn { #1 }
                {
                  (  { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
                  [  { \dim_sub:Nn \l_tmpa_dim { 0.2 mm } }
                  \{ { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
                }
              \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }
```

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

```
              \dim_set:Nn \l_tmpb_dim
                { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
              \str_case:nn { #2 }
                {
                  )  { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
                  ] { \dim_add:Nn \l_tmpb_dim { 0.2 mm } }
                  \} { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
                }
              \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
              \pgfusepathqstroke
              \group_end:
            }
            { \@@_error:nnn { Wrong~line~in~SubMatrix } { horizontal } { ##1 } }
        }
```

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).

```
    \str_if_empty:NF \l_@@_submatrix_name_str
      {
        \@@_pgf_rect_node:nnnnn \l_@@_submatrix_name_str
          \l_@@_x_initial_dim \l_@@_y_initial_dim
          \l_@@_x_final_dim \l_@@_y_final_dim
      }
    \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`.

```
    \begin { pgfscope }
    \pgftransformshift
      {
        \pgfpoint
          { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
          { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
      }
    \str_if_empty:NTF \l_@@_submatrix_name_str
      { \@@_node_left:nn #1 { } }
      { \@@_node_left:nn #1 { \@@_env: - \l_@@_submatrix_name_str - left } }
    \end { pgfscope }
```

Now, we deal with the right delimiter.

```
    \pgftransformshift
      {
        \pgfpoint
          { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
          { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
      }
    \str_if_empty:NTF \l_@@_submatrix_name_str
      { \@@_node_right:nnnn #2 { } { #3 } { #4 } }
      {
        \@@_node_right:nnnn #2
          { \@@_env: - \l_@@_submatrix_name_str - right } { #3 } { #4 }
      }
```

Now, we deal with the key `code` of `\SubMatrix`. That key should contain a TikZ instruction and the nodes in that instruction will be relative to the current `\SubMatrix`. That's why we need a redefinition of `\pgfpointanchor`.

```
    \cs_set_eq:NN \pgfpointanchor \@@_pgfpointanchor:n
    \flag_clear_new:N \l_@@_code_flag
    \l_@@_code_tl
  }
```

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

```
\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.
The original command `\pgfpointanchor` takes in two arguments: the name of the name and the name of the anchor. However, you don't have to modify the anchor, and that's why we do a redefinition of `\pgfpointanchor` by curryfication.

```
\cs_new:Npn \@@_pgfpointanchor:n #1
  { \exp_args:Ne \@@_old_pgfpointanchor: { \@@_pgfpointanchor_i:n { #1 } } }
```

First, we must detect whether the argument is of the form `\tikz@pp@name{...}` (the command `\tikz@pp@name` is a command of TikZ that adds the prefix and the suffix of the name. If the name refers to a TikZ node which does not exist, there isn't the wrapper `\tikz@pp@name`.

```
\cs_new:Npn \@@_pgfpointanchor_i:n #1
  { \@@_pgfpointanchor_ii:w #1 \tikz@pp@name \q_stop }
\cs_new:Npn \@@_pgfpointanchor_ii:w #1 \tikz@pp@name #2 \q_stop
  {
```

The command `\str_if_empty:nTF` is "fully expandable".

```
    \str_if_empty:nTF { #1 }
```

First, when the name of the name begins with `\tikz@pp@name`.

```
      { \@@_pgfpointanchor_iv:w #2 }
```

And now, when there is no `\tikz@pp@name`.

```
      { \@@_pgfpointanchor_ii:n { #1 } }
  }
```

In the case where the name begins with `\tikz@pp@name`, we must retrieve the second `\tikz@pp@name`, that is to say to marker that we have added at the end (cf. `\@@_pgfpointanchor_i:n`).

```
\cs_new:Npn \@@_pgfpointanchor_iv:w #1 \tikz@pp@name
  { \@@_pgfpointanchor_ii:n { #1 } }
```

With the command `\@@_pgfpointanchor_ii:n`, we deal with the actual name of the node (without the `\tikz@pp@name`). First, we have to detect whether it is of the form `i` of the form `i-j` (with an hyphen).
Remark: It would be possible to test the presence of the hyphen in an expandable way to using `\etl_if_in:nnTF` of the package etl but, as of now, we do not load etl.

```
\cs_new:Npn \@@_pgfpointanchor_ii:n #1 { \@@_pgfpointanchor_i:w #1- \q_stop }
```

```
\cs_new:Npn \@@_pgfpointanchor_i:w #1-#2 \q_stop
  {
```

The command `\str_if_empty:nTF` is "fully expandable".

```
    \str_if_empty:nTF { #2 }
```

First the case where the argument does *not* contain an hyphen.

```
      { \@@_pgfpointanchor_iii:n { #1 } }
```

And now the case the argument contains a hyphen. In that case, we have a weird construction because we must retreive the extra hyphen we have added as marker (cf. `\@@_pgfpointanchor_ii:n`).

```
      { \@@_pgfpointanchor_iii:w { #1 } #2 }
  }
```

The following function is for the case when the name contains an hyphen.

```
\cs_new:Npn \@@_pgfpointanchor_iii:w #1 #2 -
  {
```

We have to add the prefix `\@@_env:` "by hand" since we have retreived the potential `\tikz@pp@name`.

```
    \@@_env:
    - \int_eval:n { #1 + \l_@@_first_i_tl - 1 }
    - \int_eval:n { #2 + \l_@@_first_j_tl - 1 }
  }
```

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.

```
\tl_const:Nn \c_@@_integers_alist_tl
  {
    { 1 } { } { 2 } { } { 3 } { } { 4 } { } { 5 } { }
    { 6 } { } { 7 } { } { 8 } { } { 9 } { } { 10 } { }
    { 11 } { } { 12 } { } { 13 } { } { 14 } { } { 15 } { }
    { 16 } { } { 17 } { } { 18 } { } { 19 } { } { 20 } { }
  }
```

```
\cs_new:Npn \@@_pgfpointanchor_iii:n #1
  {
```

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 $i$`-|`$j$. That special form is the reason of the special form of the argument of `\pgfpointanchor` which arises witht its command `\name_of_command` (see above).
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`.

```
    \str_case:nVTF { #1 } \c_@@_integers_alist_tl
      {
        \flag_raise:N \l_@@_code_flag
```

We have to add the prefix `\@@_env:` "by hand" since we have retreived the potential `\tikz@pp@name`.

```
        \@@_env: -
        \int_if_even:nTF { \flag_height:N \l_@@_code_flag }
          { \int_eval:n { #1 + \l_@@_first_i_tl - 1 } }
          { \int_eval:n { #1 + \l_@@_first_j_tl - 1 } }
      }
      {
        \str_if_eq:eeTF { #1 } { last }
          {
            \flag_raise:N \l_@@_code_flag
            \@@_env: -
            \int_if_even:nTF { \flag_height:N \l_@@_code_flag }
              { \int_eval:n { \l_@@_last_i_tl + 1 } }
              { \int_eval:n { \l_@@_last_j_tl + 1 } }
          }
          { #1 }
      }
  }
```

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`).

```
\cs_new_protected:Npn \@@_node_left:nn #1 #2
  {
    \pgfnode
      { rectangle }
      { east }
      {
        \nullfont
        $ % $
        \@@_color:o \l_@@_delimiters_color_tl
        \left #1
        \vcenter
          {
            \nullfont
            \hrule \@height \l_tmpa_dim
                   \@depth \c_zero_dim
```

```
                    \@width \c_zero_dim
          }
        \right .
        $ % $
      }
      { #2 }
      { }
  }
```

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.

```
\cs_new_protected:Npn \@@_node_right:nnnn #1 #2 #3 #4
  {
    \pgfnode
      { rectangle }
      { west }
      {
        \nullfont
        $ % $
        \colorlet { current-color } { . }
        \@@_color:o \l_@@_delimiters_color_tl
        \left .
        \vcenter
          {
            \nullfont
            \hrule \@height \l_tmpa_dim
                   \@depth \c_zero_dim
                   \@width \c_zero_dim
          }
        \right #1
        \tl_if_empty:nF { #3 } { _ { \smash { #3 } } }
        ^ { \color { current-color } \smash { #4 } }
        $ % $
      }
      { #2 }
      { }
  }
```

## 34 Les commandes \UnderBrace et \OverBrace

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

```
\NewDocumentCommand \@@_UnderBrace { O { } m m m O { } }
  {
    \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { under }
    \ignorespaces
  }
\NewDocumentCommand \@@_OverBrace { O { } m m m O { } }
  {
    \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { over }
    \ignorespaces
  }

\keys_define:nn { nicematrix / Brace }
  {
    left-shorten .bool_set:N = \l_@@_brace_left_shorten_bool ,
    left-shorten .default:n = true ,
    left-shorten .value_forbidden:n = true ,
    right-shorten .bool_set:N = \l_@@_brace_right_shorten_bool ,
    right-shorten .default:n = true ,
    right-shorten .value_forbidden:n = true ,
    shorten .meta:n = { left-shorten , right-shorten } ,
    shorten .value_forbidden:n = true ,
    yshift .dim_set:N = \l_@@_brace_yshift_dim ,
    yshift .value_required:n = true ,
    yshift .initial:n = \c_zero_dim ,
    color .tl_set:N = \l_tmpa_tl ,
    color .value_required:n = true ,
    unknown .code:n =
      \@@_unknown_key:nn
        { nicematrix / Brace }
        { Unknown~key~for~Brace }
  }
```

#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.

```
\cs_new_protected:Npn \@@_brace:nnnnn #1 #2 #3 #4 #5
  {
    \group_begin:
```

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

```
    \@@_compute_i_j:nn { #1 } { #2 }
    \bool_lazy_or:nnTF
      { \int_compare_p:nNn { \l_@@_last_i_tl } > { \g_@@_row_total_int } }
      { \int_compare_p:nNn { \l_@@_last_j_tl } > { \g_@@_col_total_int } }
      {
        \str_if_eq:eeTF { #5 } { under }
          { \@@_error:nn { Construct~too~large } { \UnderBrace } }
          { \@@_error:nn { Construct~too~large } { \OverBrace } }
      }
      {
        \tl_clear:N \l_tmpa_tl
        \keys_set:nn { nicematrix / Brace } { #4 }
        \tl_if_empty:NF \l_tmpa_tl { \color { \l_tmpa_tl } }
        \pgfpicture
        \pgfrememberpicturepositiononpagetrue
        \pgf@relevantforpicturesizefalse
        \bool_if:NT \l_@@_brace_left_shorten_bool
          {
            \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
            \int_step_inline:nnn { \l_@@_first_i_tl } { \l_@@_last_i_tl }
              {
                \cs_if_exist:cT
                  { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
                  {
                    \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }

                    \dim_compare:nNnT { \pgf@x } < { \l_@@_x_initial_dim }
                      { \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x }
                  }
              }
          }
        \bool_lazy_or:nnT
          { \bool_not_p:n \l_@@_brace_left_shorten_bool }
          { \dim_compare_p:nNn { \l_@@_x_initial_dim } = { \c_max_dim } }
          {
            \@@_qpoint:n { col - \l_@@_first_j_tl }
            \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
          }
        \bool_if:NT \l_@@_brace_right_shorten_bool
          {
            \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
            \int_step_inline:nnn { \l_@@_first_i_tl } { \l_@@_last_i_tl }
              {
                \cs_if_exist:cT
                  { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
                  {
                    \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
                    \dim_compare:nNnT { \pgf@x } > { \l_@@_x_final_dim }
                      { \dim_set_eq:NN \l_@@_x_final_dim \pgf@x }
                  }
              }
          }
        \bool_lazy_or:nnT
          { \bool_not_p:n \l_@@_brace_right_shorten_bool }
          { \dim_compare_p:nNn { \l_@@_x_final_dim } = { - \c_max_dim } }
          {
            \@@_qpoint:n { col - \int_eval:n { \l_@@_last_j_tl + 1 } }
            \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
          }
        \pgfset { inner~sep = \c_zero_dim }
        \str_if_eq:eeTF { #5 } { under }
          { \@@_underbrace_i:n { #3 } }
          { \@@_overbrace_i:n { #3 } }
        \endpgfpicture
      }
    \group_end:
  }
```

The argument is the text to put above the brace.

```
\cs_new_protected:Npn \@@_overbrace_i:n #1
  {
    \@@_qpoint:n { row - \l_@@_first_i_tl }
    \pgftransformshift
      {
        \pgfpoint
          { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
          { \pgf@y + \l_@@_brace_yshift_dim - 3 pt }
      }
    \pgfnode
      { rectangle }
      { south }
      {
        \vtop
          {
            \group_begin:
            \everycr { }
            \halign
              {
                \hfil ## \hfil \crcr
                \bool_if:NTF \l_@@_tabular_bool
                  { \begin { tabular } { c } #1 \end { tabular } }
                  { $ \begin { array } { c } #1 \end { array } $ }
                \cr
                $ % $
                \overbrace
                  {
                    \hbox_to_wd:nn
                      { \l_@@_x_final_dim - \l_@@_x_initial_dim }
                      { }
                  }
                $ % $
              \cr
              }
            \group_end:
          }
      }
      { }
      { }
  }
```

The argument is the text to put under the brace.

```
\cs_new_protected:Npn \@@_underbrace_i:n #1
  {
    \@@_qpoint:n { row - \int_eval:n { \l_@@_last_i_tl + 1 } }
    \pgftransformshift
      {
        \pgfpoint
          { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
          { \pgf@y  - \l_@@_brace_yshift_dim + 3 pt }
      }
    \pgfnode
      { rectangle }
      { north }
      {
        \group_begin:
        \everycr { }
        \vbox
          {
            \halign
              {
                \hfil ## \hfil \crcr
                $ % $
                \underbrace
                  {
                    \hbox_to_wd:nn
                      { \l_@@_x_final_dim - \l_@@_x_initial_dim }
                      { }
                  }
                $ % $
                \cr
                \bool_if:NTF \l_@@_tabular_bool
                  { \begin { tabular } { c } #1 \end { tabular } }
                  { $ \begin { array } { c } #1 \end { array } $ }
                \cr
              }
          }
        \group_end:
      }
      { }
      { }
  }
```

## 35 The commands HBrace et VBrace

The TikZ style `nicematrix/brace` is a TikZ style used to draw the braces created by `\Hbrace` and `\Vbrace`.
We can't load that definition right away because of course, maybe the final user has not yet loaded TikZ (`\Hbrace` and `\Vbrace` are available only when TikZ is loaded and also its library `decorations.pathreplacing`).

```
\AddToHook { package / tikz / after }
  {
    \tikzset
      {
        nicematrix / brace / .style =
          {
            decoration = { brace , raise = -0.15 em } ,
            decorate ,
          } ,
```

Unlike the previous one, the following set of keys is internal. It won't be provided by the final user.

```
        nicematrix / mirrored-brace / .style =
          {
            nicematrix / brace ,
            decoration = mirror ,
          }
      }
  }
```

The following set of keys will be used only for security since the keys will be sent to the command `\Ldots` or `\Vdots`.

```
\keys_define:nn { nicematrix / Hbrace }
  {
    color .code:n = ,
    horizontal-label .code:n = ,
    horizontal-labels .code:n = ,
    shorten .code:n = ,
    shorten-start .code:n = ,
    shorten-end .code:n = ,
    shorten+ .code:n = ,
    shorten-start+ .code:n = ,
    shorten-end+ .code:n = ,
    shorten~+ .code:n = ,
    shorten-start~+ .code:n = ,
    shorten-end~+ .code:n = ,
    brace-shift .code:n = ,
    brace-shift+ .code:n = ,
    brace-shift~+ .code:n = ,
    unknown .code:n = \@@_fatal:n { Unknown~key~for~Hbrace }
  }
```

Here we need an "fully expandable" command.

```
\NewExpandableDocumentCommand { \@@_Hbrace } { O { } m m }
  {
    \cs_if_exist:cTF { tikz@library@decorations.pathreplacing@loaded }
      { \@@_hbrace:nnn { #1 } { #2 } { #3 } }
      { \@@_error:nn { Hbrace~not~allowed } { \Hbrace } }
  }
```

The following command must *not* be protected because of the `\Hdotsfor` which contains a `\multicolumn` (whereas the similar command `\@@_vbrace:nnn` *must* be protected).

```
\cs_new:Npn \@@_hbrace:nnn #1 #2 #3
  {
    \int_compare:nNnTF { \c@iRow } < { 2 }
      {
```

We recall that `\str_if_eq:nnTF` is "fully expandable".

```
        \str_if_eq:nnTF { #2 } { * }
          {
            \bool_set_true:N \l_@@_nullify_dots_bool
            \Ldots
              [
                line-style = nicematrix / brace ,
                #1 ,
                up =
                  \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
              ]
          }
          {
            \Hdotsfor
              [
                line-style = nicematrix / brace ,
                #1 ,
                up =
                  \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
              ]
              { #2 }
          }
      }
      {
        \str_if_eq:nnTF { #2 } { * }
          {
            \bool_set_true:N \l_@@_nullify_dots_bool
            \Ldots
              [
                line-style = nicematrix / mirrored-brace ,
                #1 ,
                down =
                  \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
              ]
          }
          {
            \Hdotsfor
              [
                line-style = nicematrix / mirrored-brace ,
                #1 ,
                down =
                  \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
              ]
            { #2 }
          }
      }
   \keys_set:nn { nicematrix / Hbrace } { #1 }
  }
```

```
\NewDocumentCommand { \@@_Vbrace } { O { } m m }
  {
    \cs_if_exist:cTF { tikz@library@decorations.pathreplacing@loaded }
      { \@@_vbrace:nnn { #1 } { #2 } { #3 } }
      { \@@_error:nn { Hbrace~not~allowed } { \Vbrace } }
  }
```

The following command must be protected (whereas the similar command `\@@_hbrace:nnn` must not.

```
\cs_new_protected:Npn \@@_vbrace:nnn #1 #2 #3
  {
    \int_compare:nNnTF { \c@jCol } < { 2 }
      {
        \str_if_eq:nnTF { #2 } { * }
          {
            \bool_set_true:N \l_@@_nullify_dots_bool
            \Vdots
              [
                Vbrace ,
                line-style = nicematrix / mirrored-brace ,
                #1 ,
                down =
                  \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
              ]
          }
          {
            \Vdotsfor
              [
                Vbrace ,
                line-style = nicematrix / mirrored-brace ,
                #1 ,
                down =
                  \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
              ]
            { #2 }
          }
      }
      {
        \str_if_eq:nnTF { #2 } { * }
          {
            \bool_set_true:N \l_@@_nullify_dots_bool
            \Vdots
              [
                Vbrace ,
                line-style =  nicematrix / brace ,
                #1 ,
                up =
                  \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
              ]
          }
          {
            \Vdotsfor
              [
                Vbrace ,
                line-style = nicematrix / brace ,
                #1 ,
                up =
                  \bool_if:NT \l_@@_tabular_bool \text { \exp_not:n { #3 } }
              ]
            { #2 }
          }
      }
   \keys_set:nn { nicematrix / Hbrace } { #1 }
  }
```

## 36 The command TikzEveryCell

```
\bool_new:N \l_@@_not_empty_bool
\bool_new:N \l_@@_empty_bool

\keys_define:nn { nicematrix / TikzEveryCell }
  {
    not-empty .code:n =
      \bool_lazy_or:nnTF
        { \l_@@_in_code_after_bool }
        { \g_@@_create_cell_nodes_bool }
        { \bool_set_true:N \l_@@_not_empty_bool }
        { \@@_error:n { detection~of~empty~cells } } ,
    not-empty .value_forbidden:n = true ,
    empty .code:n =
      \bool_lazy_or:nnTF
```

```
        { \l_@@_in_code_after_bool }
        { \g_@@_create_cell_nodes_bool }
        { \bool_set_true:N \l_@@_empty_bool }
        { \@@_error:n { detection~of~empty~cells } } ,
    empty .value_forbidden:n = true ,
    unknown .code:n = \@@_error:n { Unknown~key~for~TikzEveryCell }
  }


\NewDocumentCommand { \@@_TikzEveryCell } { O { } m }
  {
    \IfPackageLoadedTF { tikz }
      {
        \group_begin:
        \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.

```
        \tl_set:Nn \l_tmpa_tl { { #2 } }
        \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
          { \@@_for_a_block:nnnnn ##1 }
        \@@_all_the_cells:
        \group_end:
      }
      { \@@_error:n { TikzEveryCell~without~tikz } }
  }


\cs_new_protected:Nn \@@_all_the_cells:
  {
    \int_step_inline:nn \c@iRow
      {
        \int_step_inline:nn \c@jCol
          {
            \cs_if_exist:cF { cell - ##1 - ####1 }
              {
                \clist_if_in:NeF \l_@@_corners_cells_clist
                  { ##1 - ####1 }
                  {
                    \bool_set_false:N \l_tmpa_bool
                    \cs_if_exist:cTF
                      { pgf @ sh @ ns @ \@@_env: - ##1 - ####1 }
                      {
                        \bool_if:NF \l_@@_empty_bool
                          { \bool_set_true:N \l_tmpa_bool }
                      }
                      {
                        \bool_if:NF \l_@@_not_empty_bool
                          { \bool_set_true:N \l_tmpa_bool }
                      }
                    \bool_if:NT \l_tmpa_bool
                      {
                        \@@_block_tikz:onnnn
                        \l_tmpa_tl { ##1 } { ####1 } { ##1 } { ####1 }
                      }
                  }
              }
          }
      }
  }

\cs_new_protected:Nn \@@_for_a_block:nnnnn
  {
    \bool_if:NF \l_@@_empty_bool
```

```
      {
        \@@_block_tikz:onnnn
          \l_tmpa_tl { #1 } { #2 } { #3 } { #4 }
      }
    \@@_mark_cells_of_block:nnnn { #1 } { #2 } { #3 } { #4 }
  }

\cs_new_protected:Nn \@@_mark_cells_of_block:nnnn
  {
    \int_step_inline:nnn { #1 } { #3 }
      {
        \int_step_inline:nnn { #2 } { #4 }
          { \cs_set_nopar:cpn { cell - ##1 - ####1 } { } }
      }
  }
```

## 37 The command \CreateBlocksInColumn

```
\cs_new_protected:Npn \@@_create_blocks_in_col:n #1
  { \clist_gput_right:Nn \g_@@_cbic_clist { #1 } }
```

```
\cs_new_protected:Npn \@@_cbic:
  {
    \clist_map_inline:Nn \g_@@_cbic_clist
      {
        \cs_set:cpn
          {
            pgf @ sh @ ns @ \@@_env:
            - \int_eval:n { \c@iRow + 1 } - ##1 }
          { rien }
```

\l_tmpa_int will be the first row of the block being created.

```
        \int_set:Nn \l_tmpa_int { 1 }
        \int_step_inline:nn { \c@iRow + 1 }
          {
            \cs_if_exist:cT
              { pgf @ sh @ ns @ \@@_env: - ####1 - ##1 }
              {
                \int_compare:nNnT { ####1 } > { \l_tmpa_int + 1 }
                  {
                    \seq_gput_right:Ne \g_@@_pos_of_blocks_seq
                      {
                        { \int_use:N \l_tmpa_int }
                        { ##1 }
                        { \int_eval:n { ####1 - 1 } }
                        { ##1 }
                        { }
                      }
                  }
                \int_set:Nn \l_tmpa_int { ####1 }
              }
          }
      }
    \clist_gclear:N \g_@@_cbic_clist
  }
```

## 38 The command \ShowCellNames

```
\NewDocumentCommand \@@_ShowCellNames { }
 {
   \bool_if:NT \l_@@_in_code_after_bool
```

```
    {
      \pgfpicture
      \pgfrememberpicturepositiononpagetrue
      \pgf@relevantforpicturesizefalse
      \pgfpathrectanglecorners
        { \@@_qpoint:n { 1 } }
        {
          \@@_qpoint:n
            { \int_eval:n { \int_max:nn { \c@iRow } { \c@jCol } + 1 } }
        }
      \pgfsetfillopacity { 0.75 }
      \pgfsetfillcolor { white }
      \pgfusepathqfill
      \endpgfpicture
    }
  \dim_gzero_new:N \g_@@_tmpc_dim
  \dim_gzero_new:N \g_@@_tmpd_dim
  \dim_gzero_new:N \g_@@_tmpe_dim
  \int_step_inline:nn { \c@iRow }
    {
      \bool_if:NTF \l_@@_in_code_after_bool
        {
          \pgfpicture
          \pgfrememberpicturepositiononpagetrue
          \pgf@relevantforpicturesizefalse
        }
        { \begin { pgfpicture } }
      \@@_qpoint:n { row - ##1 }
      \dim_set_eq:NN \l_tmpa_dim \pgf@y
      \@@_qpoint:n { row - \int_eval:n { ##1 + 1 } }
      \dim_gset:Nn \g_tmpa_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
      \dim_gset:Nn \g_tmpb_dim { \l_tmpa_dim - \pgf@y }
      \bool_if:NTF \l_@@_in_code_after_bool
        { \endpgfpicture }
        { \end { pgfpicture } }
      \int_step_inline:nn { \c@jCol }
        {
          \hbox_set:Nn \l_tmpa_box
            {
              \normalfont \Large \sffamily \bfseries
              \bool_if:NTF \l_@@_in_code_after_bool
                { \color { red } }
                { \color { red ! 50 } }
              ##1 - ####1
            }
          \bool_if:NTF \l_@@_in_code_after_bool
            {
              \pgfpicture
              \pgfrememberpicturepositiononpagetrue
              \pgf@relevantforpicturesizefalse
            }
            { \begin { pgfpicture } }
          \@@_qpoint:n { col - ####1 }
          \dim_gset_eq:NN \g_@@_tmpc_dim \pgf@x
          \@@_qpoint:n { col - \int_eval:n { ####1 + 1 } }
          \dim_gset:Nn \g_@@_tmpd_dim { \pgf@x - \g_@@_tmpc_dim }
          \dim_gset_eq:NN \g_@@_tmpe_dim \pgf@x
          \bool_if:NTF \l_@@_in_code_after_bool
            { \endpgfpicture }
            { \end { pgfpicture } }
          \fp_set:Nn \l_tmpa_fp
            {
              \fp_min:nn
```

```
                  {
                    \fp_min:nn
                      { \dim_ratio:nn \g_@@_tmpd_dim { \box_wd:N \l_tmpa_box } }
                      { \dim_ratio:nn \g_tmpb_dim { \box_ht_plus_dp:N \l_tmpa_box } }
                  }
                  { 1.0 }
              }
            \box_scale:Nnn \l_tmpa_box { \fp_use:N \l_tmpa_fp } { \fp_use:N \l_tmpa_fp }
            \pgfpicture
            \pgfrememberpicturepositiononpagetrue
            \pgf@relevantforpicturesizefalse
            \pgftransformshift
              {
                \pgfpoint
                  { 0.5 * ( \g_@@_tmpc_dim + \g_@@_tmpe_dim ) }
                  { \dim_use:N \g_tmpa_dim }
              }
            \pgfnode
              { rectangle }
              { center }
              { \box_use:N \l_tmpa_box }
              { }
              { }
            \endpgfpicture
          }
      }
  }
```

## 39 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.

```
\bool_new:N \g_@@_footnotehyper_bool
```

The boolean `\g_@@_footnote_bool` will indicate if the option `footnote` is used, but quickly, it will also be set to `true` if the option `footnotehyper` is used.

```
\bool_new:N \g_@@_footnote_bool
\msg_new:nnnn { nicematrix } { Unknown~key~for~package }
  {
    You~have~used~the~key~' \l_keys_key_str '~when~loading~nicematrix~
    but~that~key~is~unknown. \\
    It~will~be~ignored. \\
    For~a~list~of~the~available~keys,~type~H~<return>.
  }
  {
    The~available~keys~are~(in~alphabetic~order):~
    footnote,~
    footnotehyper,~
    messages-for-Overleaf,~
    renew-dots~and~
    renew-matrix.
  }
\keys_define:nn { nicematrix }
  {
    renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
    renew-dots .value_forbidden:n = true ,
    renew-matrix .code:n = \@@_renew_matrix: ,
    renew-matrix .value_forbidden:n = true ,
    messages-for-Overleaf .bool_set:N = \g_@@_messages_for_Overleaf_bool ,
    footnote .bool_set:N = \g_@@_footnote_bool ,
    footnotehyper .bool_set:N = \g_@@_footnotehyper_bool ,
    unknown .code:n = \@@_error:n { Unknown~key~for~package }
  }
\ProcessKeyOptions
```

```
\@@_msg_new:nn { footnote~with~footnotehyper~package }
  {
    You~can't~use~the~option~'footnote'~because~the~package~
    footnotehyper~has~already~been~loaded.~
    If~you~want,~you~can~use~the~option~'footnotehyper'~and~the~footnotes~
    within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
    of~the~package~footnotehyper.\\
    The~package~footnote~won't~be~loaded.
  }
\@@_msg_new:nn { footnotehyper~with~footnote~package }
  {
    You~can't~use~the~option~'footnotehyper'~because~the~package~
    footnote~has~already~been~loaded.~
    If~you~want,~you~can~use~the~option~'footnote'~and~the~footnotes~
    within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
    of~the~package~footnote.\\
    The~package~footnotehyper~won't~be~loaded.
  }
```

```
\bool_if:NT \g_@@_footnote_bool
  {
```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```
    \IfClassLoadedTF { beamer }
      { \bool_set_false:N \g_@@_footnote_bool }
      {
        \IfPackageLoadedTF { footnotehyper }
          { \@@_error:n { footnote~with~footnotehyper~package } }
          { \usepackage { footnote } }
      }
  }
\bool_if:NT \g_@@_footnotehyper_bool
  {
```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```
    \IfClassLoadedTF { beamer }
      { \bool_set_false:N \g_@@_footnote_bool }
      {
        \IfPackageLoadedTF { footnote }
          { \@@_error:n { footnotehyper~with~footnote~package } }
          { \usepackage { footnotehyper } }
      }
    \bool_set_true:N \g_@@_footnote_bool
  }
```

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}`.

## 40 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.

```
\bool_new:N \l_@@_underscore_loaded_bool
\IfPackageLoadedT { underscore }
  { \bool_set_true:N \l_@@_underscore_loaded_bool }
```

```
\hook_gput_code:nnn { begindocument } { . }
  {
    \bool_if:NF \l_@@_underscore_loaded_bool
      {
        \IfPackageLoadedT { underscore }
          { \@@_error:n { underscore~after~nicematrix } }
      }
  }
```

## 41 Compatibility with threeparttable

```
\hook_gput_code:nnn { begindocument } { . }
  {
    \IfPackageLoadedT { threeparttable }
      {
        \AddToHook { env / threeparttable / begin }
          {
            \TPT@hookin { NiceTabular }
            \TPT@hookin { NiceTabular* }
            \TPT@hookin { NiceTabularX }
          }
      }
  }
```

## 42 Error messages of the package

When there is a unknown key, maybe the user has tried to use an inexistent "additive syntax" for that key. Of course, in that case, the last character of the name of the key is +.
#1 is a clist of names of sets of keys and #2 is the error message to send.

```
\cs_new_protected:Npn \@@_unknown_key:nn #1 #2
  {
    \str_if_eq:eeTF
      { \str_item:Nn \l_keys_key_str { \str_count:N \l_keys_key_str } }
      { + }
      {
        \str_set:Ne \l_tmpa_str
          { \str_range:Nnn \l_keys_key_str { 1 } { \str_count:N \l_keys_key_str - 1 } }
        \bool_set_false:N \l_tmpa_bool
        \clist_map_inline:nn { #1 }
          {
            \keys_if_exist:neT { ##1 } { \l_tmpa_str }
              {
                \@@_error:n { key~without~+~exists }
                \bool_set_true:N \l_tmpa_bool
                \clist_map_break:
              }
          }
        \bool_if:NF \l_tmpa_bool
          {
            \str_set:Ne \l_keys_key_str { \tl_trim_right_spaces:V \l_tmpa_str }
            \@@_unknown_key_i:nn { #1 } { #2 }
          }
      }
      { \@@_unknown_key_i:nn { #1 } { #2 } }
  }
```

We try a normalisation of the name of the key, and, when that normal form exists, we add that information in the error message.
The normal form is the lower case form of the key, with all the spaces replaced by hyphens (there is never spaces in the keys of nicematrix).
#1 is a clist of names of sets of keys and #2 is the error message to send.

```
\cs_new_protected:Npn \@@_unknown_key_i:nn #1 #2
  {
    \str_set_eq:NN \l_tmpa_str \l_keys_key_str
    \str_replace_all:Nnn \l_tmpa_str { ~ } { - }
    \str_set:Ne \l_tmpa_str { \str_lowercase:f { \l_tmpa_str } }
    \bool_set_false:N \l_tmpa_bool
    \clist_map_inline:nn { #1 }
      {
        \keys_if_exist:neT { ##1 } { \l_tmpa_str }
          {
            \@@_error:n { key~with~normal~form~exists }
            \bool_set_true:N \l_tmpa_bool
            \clist_map_break:
          }
      }
    \bool_if:NF \l_tmpa_bool
      {
        \@@_error:n { #2 }
```

If messages-for-Overleaf is not in force, the list of the available keys is not written in the main error message but only in the complement (if the final user presses H). That's why we write, in all circunstancies, the list of the available keys in order to facilitate the work of the systems which analyze the error by IA (such as Prism).

```
        \bool_if:NF \g_@@_messages_for_Overleaf_bool
          { \msg_info:nn { nicematrix } { #2~+ } }
      }
  }
\@@_msg_new:nn { key~without~+~exists }
  {
    The~key~'\tl_trim_right_spaces:V \l_tmpa_str'~exists~but~does~not~accept~an~
    additive~syntax~(with~+=).\\
    It~will~be~ignored.\\
  }
\@@_msg_new:nn { key~with~normal~form~exists }
  {
    The~key~'\l_keys_key_str'~does~not~exists.\\
    It~will~be~ignored.\\
    Maybe~you~want~to~use~the~key~'\l_tmpa_str'.
  }
\str_const:Ne \c_@@_available_keys_str
  {
    \bool_if:nT { ! \g_@@_messages_for_Overleaf_bool }
      { For~a~list~of~the~available~keys,~type~H~<return>. }
  }
```

```
\seq_new:N \g_@@_types_of_matrix_seq
\seq_gset_from_clist:Nn \g_@@_types_of_matrix_seq
  {
    NiceMatrix ,
    pNiceMatrix , bNiceMatrix , vNiceMatrix, BNiceMatrix, VNiceMatrix
  }
\seq_gset_map_e:NNn \g_@@_types_of_matrix_seq \g_@@_types_of_matrix_seq
  { \tl_to_str:n { #1 } }
```

If the user uses too much columns, the command `\@@_err_too_many_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:NoF` 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`.

```
\cs_new_protected:Npn \@@_err_too_many_cols:
  {
    \seq_if_in:NoF \g_@@_types_of_matrix_seq \g_@@_name_env_str
      { \@@_fatal:nn { too~many~cols~for~array } }
    \int_compare:nNnT { \l_@@_last_col_int } = { -2 }
      { \@@_fatal:n { too~many~cols~for~matrix } }
    \int_compare:nNnT { \l_@@_last_col_int } = { -1 }
      { \@@_fatal:n { too~many~cols~for~matrix } }
    \bool_if:NF \l_@@_last_col_without_value_bool
      { \@@_fatal:n { too~many~cols~for~matrix~with~last~col } }
  }
```

The following command must *not* be protected since it's used in an error message.

```
\cs_new:Npn \@@_message_hdotsfor:
  {
    \tl_if_empty:oF \g_@@_HVdotsfor_lines_tl
      { ~Maybe~your~use~of~ \token_to_str:N \Hdotsfor \ or~
        \token_to_str:N \Hbrace \ is~incorrect. }
  }

\cs_new_protected:Npn \@@_Hline_in_cell:
  { \@@_fatal:n { Misuse~of~Hline } }
\@@_msg_new:nn { Misuse~of~Hline }
  {
    Misuse~of~Hline. \\
    Error~in~your~row~ \int_eval:n { \c@iRow }. \\
    \token_to_str:N \Hline\ (like \token_to_str:N \hline)~must~be~used~only~
    at~the~beginning~of~a~row.\\
    That~error~is~fatal.
  }
\@@_msg_new:nn { hvlines,~rounded-corners~and~corners }
  {
    Incompatible~options.\\
    You~should~not~use~'hvlines',~'rounded-corners'~and~'corners'~at~the~same~time.\\
    The~output~will~not~be~reliable.
  }
\@@_msg_new:nn { Body~alone }
  {
    \token_to_str:N \Body\ alone. \\
    You~have~used~\token_to_str:N \Body\ without~\token_to_str:N \CodeBefore.\\
    That~error~is~fatal.
  }
\@@_msg_new:nn { cellcolor~in~Block }
  {
    Bad~use~of~\token_to_str:N \cellcolor \\
    You~can't~use~\token_to_str:N \cellcolor\ in~\token_to_str:N \Block\
    \bool_if:NTF \l_@@_amp_in_blocks_bool
      { (but~you~could~use~it~in~a~sub-block~since~'&-in-blocks'~is~in~force) }
      { (it's~possible~in~a~sub-block~when~'&-in-blocks'~is~in~force) }
    .~Here,~you~should~use~the~key~'fill'~of~the~block.\\
    That~command~will~be~ignored.
  }
\@@_msg_new:nn { rowcolor~in~Block }
  {
    Bad~use~of~\token_to_str:N \rowcolor \\
    You~can't~use~\token_to_str:N \rowcolor\ in~\token_to_str:N \Block.\\
    That~command~will~be~ignored.
  }
\@@_msg_new:nn { key~color-inside }
  {
    Deleted~key.\\
    The~key~'color-inside'~(and~its~alias~'colortbl-like')~has~been~deleted~in
    ~'nicematrix'~and~must~not~be~used.\\
    This~error~is~fatal.
  }
\@@_msg_new:nn { invalid~weight }
  {
    Unknown~key.\\
    The~key~' \l_keys_key_str '~of~your~column~X~is~unknown~and~will~be~ignored.~
    The~available~keys~are:~l,~c,~r,~t~(=p),~m,~b,~V~
    \IfPackageLoadedTF { varwidth }
      { (since~'varwidth'~is~loaded)~}
      { (if~you~load~'varwidth')~}
    and~real~numbers~for~the~weight~of~the~X~column.
  }
\@@_msg_new:nn { last~col~not~used }
  {
    Column~not~used.\\
    The~key~'last-col'~is~in~force~but~you~have~not~used~that~last~column~
    in~your~\@@_full_name_env: .~
    However,~you~can~go~on.
  }
\@@_msg_new:nn { too~many~cols~for~matrix~with~last~col }
  {
    Too~many~columns.\\
    In~the~row~ \int_eval:n { \c@iRow },~
    you~try~to~use~more~columns~
    than~allowed~by~your~ \@@_full_name_env: .
    \@@_message_hdotsfor: \
    The~maximal~number~of~columns~is~ \int_eval:n { \l_@@_last_col_int - 1 }~
    (plus~the~exterior~columns).~This~error~is~fatal.
  }
\@@_msg_new:nn { too~many~cols~for~matrix }
  {
    Too~many~columns.\\
    In~the~row~ \int_eval:n { \c@iRow } ,~
    you~try~to~use~more~columns~than~allowed~by~your~ \@@_full_name_env: .
    \@@_message_hdotsfor: \
    Recall~that~the~maximal~number~of~columns~for~a~matrix~
    (excepted~the~potential~exterior~columns)~is~fixed~by~the~
    LaTeX~counter~'MaxMatrixCols'.~
    Its~current~value~is~ \int_use:N \c@MaxMatrixCols \
    (use~ \token_to_str:N \setcounter \ to~change~that~value).~
    This~error~is~fatal.
  }

\@@_msg_new:nn { too~many~cols~for~array }
  {
    Too~many~columns.\\
    In~the~row~ \int_eval:n { \c@iRow } ,~
    ~you~try~to~use~more~columns~than~allowed~by~your~
    \@@_full_name_env: . \@@_message_hdotsfor: \ The~maximal~number~of~columns~is~
    \int_use:N \g_@@_static_num_of_col_int \
    \bool_if:nT
      { \int_compare_p:n { \l_@@_first_col_int = 0 } || \g_@@_last_col_found_bool }
      { (plus~the~exterior~ones)~}
    since~the~preamble~is~' \g_@@_user_preamble_tl '.\\
    This~error~is~fatal.
  }
\@@_msg_new:nn { columns~not~used }
  {
    Columns~not~used.\\
    The~preamble~of~your~ \@@_full_name_env: \ is~' \g_@@_user_preamble_tl '.~
    It~announces~ \int_use:N \g_@@_static_num_of_col_int \
    columns~but~you~only~used~ \int_use:N \c@jCol .\\
    The~columns~you~did~not~used~won't~be~created.\\
    You~won't~have~similar~warning~till~the~end~of~the~document.
  }
\@@_msg_new:nn { empty~preamble }
  {
    Empty~preamble.\\
    The~preamble~of~your~ \@@_full_name_env: \ is~empty.\\
    This~error~is~fatal.
  }
\@@_msg_new:nn { in~first~col }
  {
    Erroneous~use.\\
    You~can't~use~the~command~#1 in~the~first~column~(number~0)~of~the~array.\\
    That~command~will~be~ignored.\\
    You~can~try~to~delete~the~key~'first-col'.
  }
\@@_msg_new:nn { in~last~col }
  {
    Erroneous~use.\\
    You~can't~use~the~command~#1 in~the~last~column~(exterior)~of~the~array.\\
    That~command~will~be~ignored.\\
    You~can~try~to~delete~the~key~'last-col'.
  }
\@@_msg_new:nn { in~first~row }
  {
    Erroneous~use.\\
    You~can't~use~the~command~#1 in~the~first~row~(number~0)~of~the~array.\\
    That~command~will~be~ignored.\\
    You~can~try~to~delete~the~key~'first-row'.
  }
\@@_msg_new:nn { in~last~row }
  {
    Erroneous~use.\\
    You~can't~use~the~command~#1 in~the~last~row~(exterior)~of~the~array.\\
    That~command~will~be~ignored.\\
    You~can~try~to~delete~the~key~'last-row'.
  }
\@@_msg_new:nn { TopRule~without~booktabs }
  {
    Erroneous~use.\\
    You~can't~use~the~command~ #1 because~'booktabs'~is~not~loaded.\\
    That~command~will~be~ignored.
  }
\@@_msg_new:nn{ rotate~in~p~col }
```

```
  {
    \token_to_str:N \rotate\ forbidden.\\
    You~should~not~use~\token_to_str:N \rotate\ in~a~column~of~type~'p',~
    'b',~'m'\IfPackageLoadedTF { varwidth } { ,~'X'~or~'V' } { ~or~'X'}.~
    If~you~go~on,~maybe~you~won't~have~the~expected~output.
  }
\@@_msg_new:nn { TopRule~without~tikz }
  {
    Erroneous~use.\\
    You~can't~use~the~command~ #1 because~TikZ~is~not~loaded.\\
    You~should~load~TikZ~with~\token_to_str:N \usepackage \{tikz\}.\\
    \IfPackageLoadedF { booktabs }
      { You~should~also~load~'booktabs'.\\ }
    That~command~will~be~ignored.
  }
\@@_msg_new:nn { caption~outside~float }
  {
    Key~caption~forbidden.\\
    You~can't~use~the~key~'caption'~because~you~are~not~in~a~floating~
    environment~(such~as~\{table\}).~This~key~will~be~ignored.
  }
\@@_msg_new:nn { short-caption~without~caption }
  {
    You~should~not~use~the~key~'short-caption'~without~'caption'.~
    However,~your~'short-caption'~will~be~used~as~'caption'.
  }
\@@_msg_new:nn { double~closing~delimiter }
  {
    Double~delimiter.\\
    You~can't~put~a~second~closing~delimiter~"#1"~just~after~a~first~closing~
    delimiter.~This~delimiter~will~be~ignored.\\
    You~can~try~to~use~\token_to_str:N \SubMatrix\ in~the~\token_to_str:N \CodeAfter.
  }
\@@_msg_new:nn { delimiter~after~opening }
  {
    Double~delimiter.\\
    You~can't~put~a~second~delimiter~"#1"~just~after~a~first~opening~
    delimiter.~That~delimiter~will~be~ignored.
  }
\@@_msg_new:nn { bad~option~for~line-style }
  {
    Bad~line~style.\\
    Since~you~haven't~loaded~TikZ,~the~only~value~you~can~give~to~'line-style'~
    is~'standard'.~That~key~will~be~ignored.\\
    You~can~load~TikZ~with~\token_to_str:N \usepackage \{tikz\}.\\
  }
\@@_msg_new:nn { corners~with~no-cell-nodes }
  {
    Incompatible~keys.\\
    You~can't~use~the~key~'corners'~here~because~the~key~'no-cell-nodes'~
    is~in~force~(you~should~deactive~the~key~'no-cell-nodes'~whose~only~goal~
    is~to~speed-up~compilation).\\
    If~you~go~on,~that~key~will~be~ignored.
  }
\@@_msg_new:nn { extra-nodes~with~no-cell-nodes }
  {
    Incompatible~keys.\\
    You~can't~create~'extra~nodes'~here~because~the~key~'no-cell-nodes'~
    is~in~force~(you~should~deactive~the~key~'no-cell-nodes'~whose~only~goal~
    is~to~speed-up~compilation).\\
    If~you~go~on,~those~extra~nodes~won't~be~created.
  }
\@@_msg_new:nn { Identical~notes~in~caption }
  {
    Identical~tabular~notes.\\
    You~can't~put~several~notes~with~the~same~content~in~
    \token_to_str:N \caption \ (but~it's~possible~in~the~main~tabular).\\
    If~you~go~on,~the~output~will~probably~be~erroneous.
  }
\@@_msg_new:nn { tabularnote~below~the~tabular }
  {
    \token_to_str:N \tabularnote \ forbidden\\
    You~can't~use~ \token_to_str:N \tabularnote \ in~the~caption~
    of~your~tabular~because~the~caption~will~be~composed~below~
    the~tabular.~If~you~want~the~caption~above~the~tabular~use~the~
    key~'caption-above'~in~ \token_to_str:N \NiceMatrixOptions .\\
    Your~ \token_to_str:N \tabularnote \ will~be~discarded~and~
    no~similar~error~will~raised~in~this~document.
  }
\@@_msg_new:nn { Unknown~key~for~rules }
  {
    Unknown~key.\\
    There~is~only~two~keys~available~here:~width~and~color.\\
    Your~key~' \l_keys_key_str '~will~be~ignored.
  }
\@@_msg_new:nn { Unknown~key~for~Hbrace }
  {
    Unknown~key.\\
    You~have~used~the~key~' \l_keys_key_str '~but~the~only~
    keys~allowed~for~the~commands~ \token_to_str:N \Hbrace \
    and~ \token_to_str:N \Vbrace \ are:~'brace-shift(+)',~'color',~
    'horizontal-label(s)',~'shorten'~'shorten-end'~
    and~'shorten-start'.\\
    That~error~is~fatal.
  }
\@@_msg_new:nn { Unknown~key~for~TikzEveryCell }
  {
    Unknown~key.\\
    There~is~only~two~keys~available~here:~
    'empty'~and~'not-empty'.\\
    Your~key~' \l_keys_key_str '~will~be~ignored.
  }
\@@_msg_new:nn { Unknown~key~for~rotate }
  {
    Unknown~key.\\
    The~only~key~available~here~is~'c'.\\
    Your~key~' \l_keys_key_str '~will~be~ignored.
  }
\@@_msg_new:nnn { Unknown~key~for~custom-line }
  {
    Unknown~key.\\
    The~key~' \l_keys_key_str '~is~unknown~in~a~'custom-line'.~
    It~you~go~on,~you~will~probably~have~other~errors. \\
    \c_@@_available_keys_str
  }
  {
    The~available~keys~are~(in~alphabetic~order):~
    ccommand,~
    color,~
    command,~
    dotted,~
    letter,~
    multiplicity,~
    sep-color,~
    tikz,~and~total-width.
  }
\@@_msg_new:nnn { Unknown~key~for~xdots }
  {
    Unknown~key.\\
    The~key~' \l_keys_key_str '~is~unknown~for~a~command~for~drawing~dotted~rules.\\
    \c_@@_available_keys_str
  }
  {
    The~available~keys~are~(in~alphabetic~order):~
    'color',~
    'horizontal(s)-labels',~
    'inter',~
    'line-style',~
    'nullify',~
    'radius',~
    'shorten',~
    'shorten-end'~and~'shorten-start'.
  }
\@@_msg_new:nn { Unknown~key~for~rowcolors }
  {
    Unknown~key.\\
    As~for~now,~there~is~only~two~keys~available~here:~'cols'~and~'respect-blocks'~
    (and~you~try~to~use~' \l_keys_key_str ')\\
    That~key~will~be~ignored.
  }
\@@_msg_new:nn { label~without~caption }
  {
    You~can't~use~the~key~'label'~in~your~\{NiceTabular\}~because~
    you~have~not~used~the~key~'caption'.~The~key~'label'~will~be~ignored.
  }
\@@_msg_new:nn { W~warning }
  {
    Line~ \msg_line_number: .~The~cell~is~too~wide~for~your~column~'W'~
    (row~ \int_use:N \c@iRow ).
  }
\@@_msg_new:nn { Construct~too~large }
  {
    Construct~too~large.\\
    Your~command~ \token_to_str:N #1
    can't~be~drawn~because~your~matrix~is~too~small.\\
    That~command~will~be~ignored.
  }
\@@_msg_new:nn { underscore~after~nicematrix }
  {
    Problem~with~'underscore'.\\
    The~package~'underscore'~should~be~loaded~before~'nicematrix'.~
    You~can~go~on~but~you~won't~be~able~to~write~something~such~as:\\
    ' \token_to_str:N \Cdots \token_to_str:N _
    \{ n \token_to_str:N \text \{ ~times \} \}'.
  }
\@@_msg_new:nn { ampersand~in~light-syntax }
  {
    Ampersand~forbidden.\\
    You~can't~use~an~ampersand~( \token_to_str:N &)~to~separate~columns~because~
    ~the~key~'light-syntax'~is~in~force.~This~error~is~fatal.
  }
```

```
\@@_msg_new:nn { double-backslash~in~light-syntax }
  {
    Double~backslash~forbidden.\\
    You~can't~use~ \token_to_str:N \\
    ~to~separate~rows~because~the~key~'light-syntax'~
    is~in~force.~You~must~use~the~character~' \l_@@_end_of_row_tl '~
    (set~by~the~key~'end-of-row').~This~error~is~fatal.
  }
\@@_msg_new:nn { hlines~with~color }
  {
    Incompatible~keys.\\
    You~can't~use~the~keys~'hlines',~'vlines'~or~'hvlines'~for~a~
    \token_to_str:N \Block \ when~the~key~'color'~or~'draw'~is~used.\\
    However,~you~can~put~several~commands~ \token_to_str:N \Block.\\
    Your~key~will~be~discarded.
  }
\@@_msg_new:nn { bad~value~for~baseline }
  {
    Bad~value~for~baseline.\\
    The~value~given~to~'baseline'~( \int_use:N \l_tmpa_int )~is~not~
    valid.~The~value~must~be~between~\int_use:N \l_@@_first_row_int\ and~
    \int_use:N \g_@@_row_total_int \ or~equal~to~'t',~'c'~or~'b'~or~of~
    the~form~'line-i'.\\
    A~value~of~1~will~be~used.
  }
\@@_msg_new:nn { bad~value~for~baseline-line }
  {
    Bad~value~for~baseline~with~line.\\
    The~value~given~to~'baseline'~( \int_use:N \l_tmpa_int )~is~not~
    valid.~The~number~of~the~line~must~be~between~1~and~
    \int_eval:n { \c@iRow + 1 } \\
    A~value~of~'line-1'~will~be~used.
  }
\@@_msg_new:nn { detection~of~empty~cells }
  {
    Problem~with~'not-empty'\\
    For~technical~reasons,~you~must~activate~
    'create-cell-nodes'~in~ \token_to_str:N \CodeBefore \
    in~order~to~use~the~key~' \l_keys_key_str '.\\
    That~key~will~be~ignored.
  }
\@@_msg_new:nn { siunitx~not~loaded }
  {
    siunitx~not~loaded\\
    You~can't~use~the~columns~'S'~because~'siunitx'~is~not~loaded.\\
    That~error~is~fatal.\\
    You~can~load~'siunitx'~with~\token_to_str:N \usepackage \{siunitx\}.\\
  }
\@@_msg_new:nn { Invalid~name }
  {
    Invalid~name.\\
    You~can't~give~the~name~' \l_keys_value_tl '~to~a~ \token_to_str:N
    \SubMatrix \ of~your~ \@@_full_name_env: .\\
    A~name~must~be~accepted~by~the~regular~expression~[A-Za-z][A-Za-z0-9]*.\\
    This~key~will~be~ignored.
  }
\@@_msg_new:nn { Hbrace~not~allowed }
  {
    Command~not~allowed.\\
    You~can't~use~the~command~ \token_to_str:N #1
    because~you~have~not~loaded~
    \IfPackageLoadedTF { tikz }
      { the~TikZ~library~'decorations.pathreplacing'.~Use~ }
      { TikZ.~ Use:~ \token_to_str:N \usepackage \{tikz\}~and~ }
    \token_to_str:N \usetikzlibrary \{decorations.pathreplacing\}. \\
    That~command~will~be~ignored.
  }
\@@_msg_new:nn { Vbrace~not~allowed }
  {
    Command~not~allowed.\\
    You~can't~use~the~command~ \token_to_str:N \Vbrace \
    because~you~have~not~loaded~TikZ~
    and~the~TikZ~library~'decorations.pathreplacing'.\\
    Use: ~\token_to_str:N \usepackage \{tikz\}~
    \token_to_str:N \usetikzlibrary \{decorations.pathreplacing\} \\
    That~command~will~be~ignored.
  }
\@@_msg_new:nn { Wrong~line~in~SubMatrix }
  {
    Wrong~line.\\
    You~try~to~draw~a~#1~line~of~number~'#2'~in~a~
    \token_to_str:N \SubMatrix \ of~your~ \@@_full_name_env: \ but~that~
    number~is~not~valid.~It~will~be~ignored.
  }
\@@_msg_new:nn { Impossible~delimiter }
  {
    Impossible~delimiter.\\
    It's~impossible~to~draw~the~#1~delimiter~of~your~
    \token_to_str:N \SubMatrix \ because~all~the~cells~are~empty~
    in~that~column.
    \bool_if:NT \l_@@_submatrix_slim_bool
      { ~Maybe~you~should~try~without~the~key~'slim'. } \\
    This~ \token_to_str:N \SubMatrix \ will~be~ignored.
  }
\@@_msg_new:nnn { width~without~X~columns }
  {
    You~have~used~the~key~'width'~but~you~have~put~no~'X'~column~in~
   the~preamble~(' \g_@@_user_preamble_tl ')~of~your~ \@@_full_name_env: .\\
    That~key~will~be~ignored.
  }
  {
    This~message~is~the~message~'width~without~X~columns'~
    of~the~module~'nicematrix'.~
    The~experimented~users~can~disable~that~message~with~
    \token_to_str:N \msg_redirect_name:nnn .\\
  }

\@@_msg_new:nn { key~multiplicity~with~dotted }
  {
    Incompatible~keys. \\
    You~have~used~the~key~'multiplicity'~with~the~key~'dotted'~
    in~a~'custom-line'.~They~are~incompatible. \\
    The~key~'multiplicity'~will~be~discarded.
  }
\@@_msg_new:nn { empty~environment }
  {
    Empty~environment.\\
    Your~ \@@_full_name_env: \ is~empty.~This~error~is~fatal.
  }
\@@_msg_new:nn { No~letter~and~no~command }
  {
    Erroneous~use.\\
    Your~use~of~'custom-line'~is~no-op~since~you~don't~have~used~the~
    key~'letter'~(for~a~letter~for~vertical~rules)~nor~the~keys~'command'~or~
    ~'ccommand'~(to~draw~horizontal~rules).\\
    However,~you~can~go~on.
  }
\@@_msg_new:nn { Forbidden~letter }
  {
    Forbidden~letter.\\
    You~can't~use~the~letter~'#1'~for~a~customized~line.~
    It~will~be~ignored.\\
    The~forbidden~letters~are:~\c_@@_forbidden_letters_str
  }
\@@_msg_new:nn { Several~letters }
  {
    Wrong~name.\\
    You~must~use~only~one~letter~as~value~for~the~key~'letter'~(and~you~
    have~used~' \l_@@_letter_str ').\\
    It~will~be~ignored.
  }
\@@_msg_new:nn { Delimiter~with~small }
  {
    Delimiter~forbidden.\\
    You~can't~put~a~delimiter~in~the~preamble~of~your~
    \@@_full_name_env: \
    because~the~key~'small'~is~in~force.\\
    This~error~is~fatal.
  }
\@@_msg_new:nn { unknown~cell~for~line~in~CodeAfter }
  {
    Unknown~cell.\\
    Your~command~ \token_to_str:N \line \{ #1 \} \{ #2 \}~in~
    the~ \token_to_str:N \CodeAfter \ of~your~ \@@_full_name_env: \
    can't~be~executed~because~a~cell~doesn't~exist.\\
    This~command~ \token_to_str:N \line \ will~be~ignored.
  }
\@@_msg_new:nnn { Duplicate~name~for~SubMatrix }
  {
    Duplicate~name.\\
    The~name~'#1'~is~already~used~for~a~ \token_to_str:N \SubMatrix \
    in~this~ \@@_full_name_env: .\\
    This~key~will~be~ignored.\\
    \bool_if:NF \g_@@_messages_for_Overleaf_bool
      { For~a~list~of~the~names~already~used,~type~H~<return>. }
  }
  {
    The~names~already~defined~in~this~ \@@_full_name_env: \ are:~
    \seq_use:Nnnn \g_@@_submatrix_names_seq { ~and~ } { ,~ } { ~and~ } .
  }
\@@_msg_new:nn { r~or~l~with~preamble }
  {
    Erroneous~use.\\
    You~can't~use~the~key~' \l_keys_key_str '~in~your~ \@@_full_name_env: .~
    You~must~specify~the~alignment~of~your~columns~with~the~preamble~of~
    your~ \@@_full_name_env: .\\
    This~key~will~be~ignored.
  }
\@@_msg_new:nn { Hdotsfor~in~col~0 }
  {
    Erroneous~use.\\
    You~can't~use~ \token_to_str:N \Hdotsfor\ or~\token_to_str:N \Hbrace\
    in~an~exterior~column~of~
    the~array.~This~error~is~fatal.
  }
\@@_msg_new:nn { bad~corner }
  {
    Bad~corner.\\
    #1~is~an~incorrect~specification~for~a~corner~(in~the~key~
    'corners').~The~available~values~are:~NW,~SW,~NE~and~SE.\\
    This~specification~of~corner~will~be~ignored.
  }
\@@_msg_new:nn { bad~border }
  {
    Bad~border.\\
    \l_keys_key_str \space ~is~an~incorrect~specification~for~a~border~
    (in~the~key~'borders'~of~the~command~ \token_to_str:N \Block ).~
    The~available~values~are:~left,~right,~top~and~bottom~(and~you~can~
    also~use~the~key~'tikz'
    \IfPackageLoadedF { tikz }
      { ~if~you~load~the~LaTeX~package~'tikz' } ).\\
    This~specification~of~border~will~be~ignored.
  }
\@@_msg_new:nn { TikzEveryCell~without~tikz }
  {
    TikZ~not~loaded.\\
    You~can't~use~ \token_to_str:N \TikzEveryCell \
    because~you~have~not~loaded~tikz.\\
    You~can~load~'tikz'~with~\token_to_str:N \usepackage \{tikz\}.\\
    This~command~will~be~ignored.
  }
\@@_msg_new:nn { tikz~key~without~tikz }
  {
    TikZ~not~loaded.\\
    You~can't~use~the~key~'tikz'~for~the~command~' \token_to_str:N
    \Block '~because~you~have~not~loaded~tikz.\\
    You~can~load~'tikz'~with~\token_to_str:N \usepackage \{tikz\}.\\
    This~key~will~be~ignored.
  }
\@@_msg_new:nn { Bad~argument~for~Block }
  {
    Bad~argument.\\
    The~first~mandatory~argument~of~\token_to_str:N \Block\ must~
    be~of~the~form~'i-j'~(or~totally~empty)~and~you~have~used:~
    '#1'. \\
    If~you~go~on,~the~\token_to_str:N \Block\ will~be~mono-cell~(as~if~
    the~argument~was~empty).
  }
\@@_msg_new:nn { last-col~non~empty~for~NiceArray }
  {
    Erroneous~use.\\
    In~the~ \@@_full_name_env: ,~you~must~use~the~key~
    'last-col'~without~value.\\
    However,~you~can~go~on~for~this~time~
    (the~value~' \l_keys_value_tl '~will~be~ignored).
  }
\@@_msg_new:nn { last-col~non~empty~for~NiceMatrixOptions }
  {
    Erroneous~use. \\
    In~\token_to_str:N \NiceMatrixOptions ,~you~must~use~the~key~
    'last-col'~without~value. \\
    However,~you~can~go~on~for~this~time~
    (the~value~' \l_keys_value_tl '~will~be~ignored).
  }
```

```
\@@_msg_new:nn { Block~too~large~1 }
  {
    Block~too~large. \\
    You~try~to~draw~a~block~in~the~cell~#1-#2~of~your~matrix~but~the~matrix~is~
    too~small~for~that~block. \\
    This~block~and~maybe~others~will~be~ignored.
  }
\@@_msg_new:nn { Block~too~large~2 }
  {
    Block~too~large. \\
    The~preamble~of~your~ \@@_full_name_env: \ announces~ \int_use:N
    \g_@@_static_num_of_col_int \
    columns~but~you~use~only~ \int_use:N \c@jCol \ and~that's~why~a~block~
    specified~in~the~cell~#1-#2~can't~be~drawn.~You~should~add~some~ampersands~
    (&)~at~the~end~of~the~first~row~of~your~ \@@_full_name_env: . \\
    This~block~and~maybe~others~will~be~ignored.
  }
\@@_msg_new:nn { unknown~column~type }
  {
    Bad~column~type. \\
    The~column~type~'#1'~in~your~ \@@_full_name_env: \
    is~unknown. \\
    This~error~is~fatal.
  }
\@@_msg_new:nn { unknown~column~type~multicolumn }
  {
    Bad~column~type. \\
    The~column~type~'#1'~in~the~command~\token_to_str:N \multicolumn \
    ~of~your~ \@@_full_name_env: \
    is~unknown. \\
    This~error~is~fatal.
  }
\@@_msg_new:nn { unknown~column~type~S }
  {
    Bad~column~type. \\
    The~column~type~'S'~in~your~ \@@_full_name_env: \ is~unknown. \\
    If~you~want~to~use~the~column~type~'S'~of~siunitx,~you~should~
    load~that~package. \\
    This~error~is~fatal.
  }
\@@_msg_new:nn { unknown~column~type~S~multicolumn }
  {
    Bad~column~type. \\
    The~column~type~'S'~in~the~command~\token_to_str:N \multicolumn \
    of~your~ \@@_full_name_env: \ is~unknown. \\
    If~you~want~to~use~the~column~type~'S'~of~siunitx,~you~should~
    load~that~package. \\
    This~error~is~fatal.
  }
\@@_msg_new:nn { tabularnote~forbidden }
  {
    Forbidden~command. \\
    You~can't~use~the~command~ \token_to_str:N \tabularnote \
    ~here.~This~command~is~available~only~in~
    \{NiceTabular\},~\{NiceTabular*\}~and~\{NiceTabularX\}~or~in~
    the~argument~of~a~command~\token_to_str:N \caption \ included~
    in~an~environment~\{table\}. \\
    This~command~will~be~ignored.
  }
\@@_msg_new:nn { borders~forbidden }
  {
    Forbidden~key.\\
    You~can't~use~the~key~'borders'~of~the~command~ \token_to_str:N \Block \
    because~the~option~'rounded-corners'~
    is~in~force~with~a~non-zero~value.\\
    This~key~will~be~ignored.
  }
\@@_msg_new:nn { bottomrule~without~booktabs }
  {
    booktabs~not~loaded.\\
    You~can't~use~the~key~'tabular/bottomrule'~because~you~haven't~
    loaded~'booktabs'.\\
    This~key~will~be~ignored.
  }
\@@_msg_new:nn { enumitem~not~loaded }
  {
    enumitem~not~loaded. \\
    You~can't~use~the~command~ \token_to_str:N \tabularnote \
    ~because~you~haven't~loaded~'enumitem'. \\
    All~the~commands~ \token_to_str:N \tabularnote \ will~be~
    ignored~in~the~document.
  }
\@@_msg_new:nn { tikz~without~tikz }
  {
    TikZ~not~loaded. \\
    You~can't~use~the~key~'tikz'~here~because~TikZ~is~not~
    loaded.~If~you~go~on,~that~key~will~be~ignored.
  }
\@@_msg_new:nn { tikz~in~custom-line~without~tikz }
  {
    TikZ~not~loaded. \\
    You~have~used~the~key~'tikz'~in~the~definition~of~a~
    customized~line~(with~'custom-line')~but~TikZ~is~not~loaded.~
    You~can~go~on~but~you~will~have~another~error~if~you~actually~
    use~that~custom~line.
  }
\@@_msg_new:nn { tikz~in~borders~without~tikz }
  {
    TikZ~not~loaded. \\
    You~have~used~the~key~'tikz'~in~a~key~'borders'~(of~a~
    command~' \token_to_str:N \Block ')~but~TikZ~is~not~loaded.~
    That~key~will~be~ignored.
  }
\@@_msg_new:nn { color~in~custom-line~with~tikz }
  {
    Erroneous~use.\\
    In~a~'custom-line',~you~have~used~both~'tikz'~and~'color',~
    which~is~forbidden~(you~should~use~'color'~inside~the~key~'tikz').~
    The~key~'color'~will~be~discarded.
  }
\@@_msg_new:nn { Wrong~last~row }
  {
    Wrong~number.\\
    You~have~used~'last-row= \int_use:N \l_@@_last_row_int '~but~your~
    \@@_full_name_env: \ seems~to~have~ \int_use:N \c@iRow \ rows.~
    If~you~go~on,~the~value~of~ \int_use:N \c@iRow \ will~be~used~for~
    last~row~but~you~should~correct~your~code.~You~can~avoid~this~
    problem~by~using~'last-row'~without~value~(more~compilations~
    might~be~necessary).
  }
\@@_msg_new:nn { Yet~in~env }
```

```
  {
    Nested~environments.\\
    Environments~of~nicematrix~can't~be~nested.\\
    This~error~is~fatal.
  }
\@@_msg_new:nn { Outside~math~mode }
  {
    Outside~math~mode.\\
    The~\@@_full_name_env: \ can~be~used~only~in~math~mode~
    (and~not~in~ \token_to_str:N \vcenter ).\\
    This~error~is~fatal.
  }
\@@_msg_new:nn { One~letter~allowed }
  {
    Bad~name.\\
    The~value~of~key~' \l_keys_key_str '~must~be~of~length~1~and~
    you~have~used~' \l_keys_value_tl '.\\
    It~will~be~ignored.
  }
\@@_msg_new:nn { TabularNote~in~CodeAfter }
  {
    Environment~\{TabularNote\}~forbidden.\\
    You~must~use~\{TabularNote\}~at~the~end~of~your~\{NiceTabular\}~
    but~*before*~the~ \token_to_str:N \CodeAfter . \\
    This~environment~\{TabularNote\}~will~be~ignored.
  }
\@@_msg_new:nn { varwidth~not~loaded }
  {
    varwidth~not~loaded.\\
    You~can't~use~the~column~type~'V'~because~'varwidth'~is~not~
    loaded.\\
    Your~column~will~behave~like~'p'.
  }
\@@_msg_new:nn { varwidth~not~loaded~in~X }
  {
    varwidth~not~loaded.\\
    You~can't~use~the~key~'V'~in~your~column~'X'~
    because~'varwidth'~is~not~loaded.\\
    It~will~be~ignored. \\
  }
\@@_msg_new:nnn { Unknown~key~for~RulesBis }
  {
    Unknown~key.\\
    Your~key~' \l_keys_key_str '~is~unknown~for~a~rule.\\
    \c_@@_available_keys_str
  }
  {
    The~available~keys~are~(in~alphabetic~order):~
    color,~
    dotted,~
    multiplicity,~
    sep-color,~
    tikz,~and~total-width.
  }

\@@_msg_new:nnn { Unknown~key~for~Block }
  {
    Unknown~key. \\
    The~key~' \l_keys_key_str '~is~unknown~for~the~command~
    \token_to_str:N \Block . \\
    It~will~be~ignored. \\
    \c_@@_available_keys_str
  }
  {
    The~available~keys~are~(in~alphabetic~order):~&-in-blocks,~ampersand-in-blocks,~
    b,~B,~borders,~c,~draw,~fill,~hlines,~hvlines,~l,~line-width,~name,~
    opacity,~rounded-corners,~r,~respect-arraystretch,~t,~T,~tikz,~transparent~
    and~vlines.
  }
\@@_msg_new:nnn { Unknown~key~for~Brace }
  {
    Unknown~key.\\
    The~key~' \l_keys_key_str '~is~unknown~for~the~commands~
    \token_to_str:N \UnderBrace \ and~ \token_to_str:N \OverBrace . \\
    It~will~be~ignored. \\
    \c_@@_available_keys_str
  }
  {
    The~available~keys~are~(in~alphabetic~order):~color,~left-shorten,~
    right-shorten,~shorten~(which~fixes~both~left-shorten~and~
    right-shorten)~and~yshift.
  }
\@@_msg_new:nnn { Unknown~key~for~CodeAfter }
  {
    Unknown~key.\\
    The~key~' \l_keys_key_str '~is~unknown.\\
    It~will~be~ignored. \\
    \c_@@_available_keys_str
  }
  {
    The~available~keys~are~(in~alphabetic~order):~
    delimiters/color,~
    rules~(with~the~subkeys~'color'~and~'width'),~
    sub-matrix~(several~subkeys)~
    and~xdots~(several~subkeys).~
    The~latter~is~for~the~command~ \token_to_str:N \line .
  }
\@@_msg_new:nnn { Unknown~key~for~CodeBefore }
  {
    Unknown~key.\\
    The~key~' \l_keys_key_str '~is~unknown.\\
    It~will~be~ignored. \\
    \c_@@_available_keys_str
  }
  {
    The~available~keys~are~(in~alphabetic~order):~
    create-cell-nodes,~
    delimiters/color~and~
    sub-matrix~(several~subkeys).
  }
\@@_msg_new:nnn { Unknown~key~for~SubMatrix }
  {
    Unknown~key.\\
    The~key~' \l_keys_key_str '~is~unknown.\\
    That~key~will~be~ignored. \\
    \c_@@_available_keys_str
  }
  {
    The~available~keys~are~(in~alphabetic~order):~
    'delimiters/color',~
    'extra-height',~
    'hlines',~
    'hvlines',~
    'left-xshift',~
    'name',~
    'right-xshift',~
    'rules'~(with~the~subkeys~'color'~and~'width'),~
    'slim',~
    'vlines'~and~'xshift'~(which~sets~both~'left-xshift'~
    and~'right-xshift').\\
  }
\@@_msg_new:nnn { Unknown~key~for~notes }
  {
    Unknown~key.\\
    The~key~' \l_keys_key_str '~is~unknown.\\
    That~key~will~be~ignored. \\
    \c_@@_available_keys_str
  }
  {
    The~available~keys~are~(in~alphabetic~order):~
    bottomrule,~
    code-after,~
    code-before(+),~
    detect-duplicates,~
    enumitem-keys,~
    enumitem-keys-para,~
    para,~
    label-in-list,~
    label-in-tabular~and~
    style.
  }
\@@_msg_new:nnn { Unknown~key~for~RowStyle }
  {
    Unknown~key.\\
    The~key~' \l_keys_key_str '~is~unknown~for~the~command~
    \token_to_str:N \RowStyle . \\
    That~key~will~be~ignored. \\
    \c_@@_available_keys_str
  }
  {
    The~available~keys~are~(in~alphabetic~order):~
    bold,~
    cell-space-top-limit(+),~
    cell-space-bottom-limit(+),~
    cell-space-limits(+),~
    color,~
    fill~(alias:~rowcolor),~
    nb-rows,~
    opacity~and~
    rounded-corners.
  }
\@@_msg_new:nnn { Unknown~key~for~NiceMatrixOptions }
  {
    Unknown~key.\\
    The~key~' \l_keys_key_str '~is~unknown~for~the~command~
    \token_to_str:N \NiceMatrixOptions . \\
    That~key~will~be~ignored. \\
    \c_@@_available_keys_str
  }
  {
    The~available~keys~are~(in~alphabetic~order):~
    &-in-blocks,~
    allow-duplicate-names,~
    ampersand-in-blocks,~
    caption-above,~
    cell-space-bottom-limit(+),~
    cell-space-limits(+),~
    cell-space-top-limit(+),~
    code-for-first-col(+),~
    code-for-first-row(+),~
    code-for-last-col(+),~
    code-for-last-row(+),~
    corners,~
    custom-key,~
    create-extra-nodes,~
    create-medium-nodes,~
    create-large-nodes,~
    custom-line,~
    delimiters~(several~subkeys),~
    end-of-row,~
    first-col,~
    first-row,~
    hlines,~
    hvlines,~
    hvlines-except-borders,~
    last-col,~
    last-row,~
    left-margin,~
    light-syntax,~
    light-syntax-expanded,~
    matrix/columns-type,~
    no-cell-nodes,~
    notes~(several~subkeys),~
    nullify-dots,~
    pgf-node-code,~
    renew-dots,~
    renew-matrix,~
    respect-arraystretch,~
    rounded-corners,~
    right-margin,~
    rules~(with~the~subkeys~'color'~and~'width'),~
    small,~
    sub-matrix~(several~subkeys),~
    vlines,~
    xdots~(several~subkeys).
  }
```

For '`{NiceArray}`', the set of keys is the same as for `{NiceMatrix}` excepted that there is no `l` and `r`.

```
\@@_msg_new:nnn { Unknown~key~for~NiceArray }
  {
    Unknown~key.\\
    The~key~' \l_keys_key_str '~is~unknown~for~the~environment~
    \{NiceArray\}. \\
    That~key~will~be~ignored. \\
    \c_@@_available_keys_str
  }
  {
    The~available~keys~are~(in~alphabetic~order):~
    &-in-blocks,~
    ampersand-in-blocks,~
    b,~
    baseline,~
    c,~
    cell-space-bottom-limit,~
    cell-space-limits,~
    cell-space-top-limit,~
    code-after,~
    code-for-first-col(+),~
    code-for-first-row(+),~
    code-for-last-col(+),~
    code-for-last-row(+),~
    columns-width,~
    corners,~
    create-blocks-in-col,~
    create-extra-nodes,~
    create-medium-nodes,~
    create-large-nodes,~
    extra-left-margin,~
    extra-right-margin,~
    first-col,~
    first-row,~
    hlines,~
    hvlines,~
    hvlines-except-borders,~
    last-col,~
    last-row,~
    left-margin,~
    light-syntax,~
    light-syntax-expanded,~
    name,~
    no-cell-nodes,~
    nullify-dots,~
    pgf-node-code,~
    renew-dots,~
    respect-arraystretch,~
    right-margin,~
    rounded-corners,~
    rules~(with~the~subkeys~'color'~and~'width'),~
    small,~
    t,~
    vlines,~
    xdots/color,~
    xdots/shorten-start(+),~
    xdots/shorten-end(+),~
    xdots/shorten(+)~and~
    xdots/line-style.
  }
```

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`).

```
\@@_msg_new:nnn { Unknown~key~for~NiceMatrix }
  {
    Unknown~key.\\
    The~key~' \l_keys_key_str '~is~unknown~for~the~
    \@@_full_name_env: . \\
    That~key~will~be~ignored. \\
    \c_@@_available_keys_str
  }
  {
    The~available~keys~are~(in~alphabetic~order):~
    &-in-blocks,~
    ampersand-in-blocks,~
    b,~
    baseline,~
    c,~
    cell-space-bottom-limit,~
    cell-space-limits,~
    cell-space-top-limit,~
    code-after,~
    code-for-first-col(+),~
    code-for-first-row(+),~
    code-for-last-col(+),~
    code-for-last-row(+),~
    columns-type,~
    columns-width,~
    corners,~
    create-blocks-in-col,~
    create-extra-nodes,~
    create-medium-nodes,~
    create-large-nodes,~
    extra-left-margin,~
    extra-right-margin,~
    first-col,~
    first-row,~
    hlines,~
    hvlines,~
    hvlines-except-borders,~
    l,~
    last-col,~
    last-row,~
    left-margin,~
    light-syntax,~
    light-syntax-expanded,~
    name,~
    no-cell-nodes,~
    nullify-dots,~
    pgf-node-code,~
    r,~
    renew-dots,~
    respect-arraystretch,~
    right-margin,~
    rounded-corners,~
    rules~(with~the~subkeys~'color'~and~'width'),~
    small,~
    t,~
    vlines,~
    xdots/color,~
    xdots/shorten-start(+),~
    xdots/shorten-end(+),~
    xdots/shorten(+)~and~
    xdots/line-style.
  }
\@@_msg_new:nnn { Unknown~key~for~NiceTabular }
  {
    Unknown~key.\\
    The~key~' \l_keys_key_str '~is~unknown~for~the~environment~
    \{NiceTabular\}. \\
    That~key~will~be~ignored. \\
    \c_@@_available_keys_str
  }
  {
    The~available~keys~are~(in~alphabetic~order):~
    &-in-blocks,~
    ampersand-in-blocks,~
    b,~
    baseline,~
    c,~
    caption,~
    cell-space-bottom-limit,~
    cell-space-limits,~
    cell-space-top-limit,~
    code-after,~
    code-for-first-col(+),~
    code-for-first-row(+),~
    code-for-last-col(+),~
    code-for-last-row(+),~
    columns-width,~
    corners,~
    custom-line,~
    create-blocks-in-col,~
    create-extra-nodes,~
    create-medium-nodes,~
    create-large-nodes,~
    extra-left-margin,~
    extra-right-margin,~
    first-col,~
    first-row,~
    hlines,~
    hvlines,~
    hvlines-except-borders,~
    label,~
    last-col,~
    last-row,~
    left-margin,~
    light-syntax,~
    light-syntax-expanded,~
    name,~
    no-cell-nodes,~
    notes~(several~subkeys),~
    nullify-dots,~
    pgf-node-code,~
    renew-dots,~
    respect-arraystretch,~
    right-margin,~
    rounded-corners,~
    rules~(with~the~subkeys~'color'~and~'width'),~
    short-caption,~
    t,~
    tabularnote,~
    vlines,~
    xdots/color,~
    xdots/shorten-start(+),~
    xdots/shorten-end(+),~
    xdots/shorten(+)~and~
    xdots/line-style.
  }
\@@_msg_new:nnn { Duplicate~name }
  {
    Duplicate~name.\\
    The~name~' \l_keys_value_tl '~is~already~used~and~you~shouldn't~use~
    the~same~environment~name~twice.~You~can~go~on,~but,~
    maybe,~you~will~have~incorrect~results~especially~
    if~you~use~'columns-width=auto'.~If~you~don't~want~to~see~this~
    message~again,~use~the~key~'allow-duplicate-names'~in~
    ' \token_to_str:N \NiceMatrixOptions '.\\
    \bool_if:NF \g_@@_messages_for_Overleaf_bool
      { For~a~list~of~the~names~already~used,~type~H~<return>. }
  }
  {
    The~names~already~defined~in~this~document~are:~
    \clist_use:Nnnn \g_@@_names_clist { ~and~ } { ,~ } { ~and~ } .
  }
\@@_msg_new:nn { caption-above~in~env }
  {
    The~key~'caption-above'~must~be~used~in~\token_to_str:N \NiceMatrixOptions.\\
    That~key~will~be~ignored.
  }
\@@_msg_new:nn { show-cell-names }
```

```
  {
    There~is~no~key~'show-cell-names'~in~nicematrix.\\
    You~should~use~the~command~\token_to_str:N \ShowCellNames\
    in~the~\token_to_str:N \CodeBefore\ or~the~\token_to_str:N
    \CodeAfter. \\
    That~key~will~be~ignored.
  }
\@@_msg_new:nn { Option~auto~for~columns-width }
  {
    Erroneous~use.\\
    You~can't~give~the~value~'auto'~to~the~key~'columns-width'~here.~
    That~key~will~be~ignored.
  }
\@@_msg_new:nn { NiceTabularX~without~X }
  {
    NiceTabularX~without~X.\\
    You~should~not~use~\{NiceTabularX\}~without~X~columns.\\
    However,~you~can~go~on.
  }
\@@_msg_new:nn { Preamble~forgotten }
  {
    Preamble~forgotten.\\
    You~have~probably~forgotten~the~preamble~of~your~
    \@@_full_name_env: . \\
    This~error~is~fatal.
  }
\@@_msg_new:nn { Invalid~col~number }
  {
    Invalid~column~number.\\
    A~color~instruction~in~the~ \token_to_str:N \CodeBefore \
    specifies~a~column~which~is~outside~the~array.~It~will~be~ignored.
  }
\@@_msg_new:nn { Invalid~row~number }
  {
    Invalid~row~number.\\
    A~color~instruction~in~the~ \token_to_str:N \CodeBefore \
    specifies~a~row~which~is~outside~the~array.~It~will~be~ignored.
  }
\@@_define_com:NNN p  (  )
\@@_define_com:NNN b  [  ]
\@@_define_com:NNN v  |  |
\@@_define_com:NNN V  \|  \|
\@@_define_com:NNN B  \{  \}
```

## Contents