The code of the package `nicematrix`

F. Pantigny
fpantigny@wanadoo.fr
May 29, 2024

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

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

1 Declaration of the package and packages loaded

The prefix `nicematrix` has been registred for this package.
See: http://mirrors.ctan.org/macros/latex/contrib/l3kernel/l3prefixes.pdf
<@@=nicematrix>

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

\begin{verbatim}
1 \RequirePackage{pgfcore}
2 \usepgfmodule{shapes}
\end{verbatim}

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

\begin{verbatim}
3 \RequirePackage{l3keys2e}
4 \ProvidesExplPackage
5 {nicematrix}
6 {%myfiledate}
7 {%myfileversion}
8 {%Enhanced arrays with the help of PGF/TikZ}
9 \RequirePackage { amsmath }
\end{verbatim}

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

\begin{verbatim}
9 \RequirePackage { array }
\end{verbatim}

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

\begin{verbatim}
11 \bool_const:Nn \c_@@_tagging_array_bool { \cs_if_exist_p:N \ar@ialign }
12 \bool_const:Nn \c_@@_testphase_table_bool
13 { \IfPackageLoadedTF { latex-lab-testphase-table } \c_true_bool \c_false_bool }
\end{verbatim}

*This document corresponds to the version 6.28 of `nicematrix`, at the date of 2024/05/29.
With Overleaf, by default, a document is compiled in non-stop mode. When there is an error, there is no way to the user to use the key H in order to have more information. That’s why we decide to put that piece of information (for the messages with such information) in the main part of the message when the key messages-for-Overleaf is used (at load-time).

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

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”.

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

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

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

However, if, one day, there is a change in the implementation of \texttt{array}, maybe this test will be broken (and nicematrix also).

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

```
\@@_msg_new:nn { Internal-error }
{ }
\begin{tabular}{ c > { \@@_security_test:n } c c }
text & & text
\end{tabular}
```

Here, the box \texttt{l_tmpa_box} will be used as sandbox to take our security test. This code has been modified in version 6.18 (see question 682891 on TeX StackExchange).
The following technic allows to create user commands with the ability to put an arbitrary number of \textit{list of (key=val)} after the name of the command.

\textit{Exemple}:
\begin{verbatim}
@@_collect_options:n \{ F \} \{ x=a,y=b \} \{ z=c,t=d \} \{ arg \}
\end{verbatim}

will be transformed in: \begin{verbatim} F\{x=a,y=b,z=c,t=d\}\{arg\} \end{verbatim}

Therefore, by writing: \begin{verbatim} \def\G{@@_collect_options:n{\F}} \end{verbatim}, the command \texttt{\G} takes in an arbitrary number of optional arguments between square brackets.

Be careful: that command is not “fully expandable” (because of \texttt{\peek_meaning:NTF}).

\begin{verbatim}
cs_new_protected:Npn \@@_collect_options:n \#1 \{ \peek_meaning:NTF [ \{ \@@_collect_options:nw { \#1 } \} \{ \#1 \} \} \}
\end{verbatim}

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

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

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

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

\section*{4 Technical definitions}

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

\begin{verbatim}
tl_const:Nn \c_@@_b_tl \{ b \}
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 \{ . \}
tl_const:Nn \c_@@_default_tl \{ default \}
tl_const:Nn \c_@@_star_tl \{ * \}
str_const:Nn \c_@@_r_str \{ r \}
str_const:Nn \c_@@_c_str \{ c \}
str_const:Nn \c_@@_l_str \{ l \}
str_const:Nn \c_@@_R_str \{ R \}
\end{verbatim}
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 n } { o }
\cs_generate_variant:Nn \str_lowercase:n { o }
\hook_gput_code:nnn { begindocument } { . }
\IfPackageLoadedTF { tikz }
{ 
\tl_const:Nn \c_@@_pgfortikzpicture_tl { \exp_not:N \tikzpicture }
\tl_const:Nn \c_@@_endpgfortikzpicture_tl { \exp_not:N \endtikzpicture }
}
\IfPackageLoadedTF { revtex4-1 }
{ \bool_const:Nn \c_@@_revtex_bool \c_true_bool }
\IfPackageLoadedTF { revtex4-2 }
{ \bool_const:Nn \c_@@_revtex_bool \c_true_bool }
{ 
\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 }
}
\IfClassLoadedTF { revtex4-1 }
\IfClassLoadedTF { revtex4-2 }
{ \bool_const:Nn \c_@@_revtex_bool \c_true_bool }
\IfIfFileExists { \rvtx@ifformat@geq }
{ \bool_const:Nn \c_@@_revtex_bool \c_true_bool }
{ \bool_const:Nn \c_@@_revtex_bool \c_false_bool }

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

\cs_if_exist:NT \rvtx@if_format@eq
{ \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:
{ 
\iou_nov:N \@@_provide_pgfsyspdfmark:
{ 
\ExpSyntaxOn
\cs_if_free:NT \pgfsyspdfmark
{ \cs_set_eq:NN \pgfsyspdfmark \@gobblethree }
}
We define a command \iddots similar to \ddots (\ldots) but with dots going forward (\ldots). 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 { \text{mkern:} 1 \mu
\boxmoveup:nn { 1 pt } { \hbox { . } } \text{mkern:} 2 \mu
\boxmoveup:nn { 4 pt } { \hbox { . } } \text{mkern:} 2 \mu
\boxmoveup:nn { 7 pt } { \vbox:n \kern 7 pt \hbox { . } } \text{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.

\checkgputcode:nnn { begindocument } { . }
{ \IfPackageLoadedTF { booktabs }
\{ \iownow:Nn \@mainaux \nicematrix@redefine@check@rerun \}
\} }

\cssetprotected:Npn \nicematrix@redefine@check@rerun
{ \cssetprotected:Npn \pgfutil@check@rerun \#1 \#2
{ \strifeq:eeF { nm- } \tlrange:nnn { \#1 } 1 3
\{ \@@oldpgfutil@check@rerun \#1 \#2 \}
\}
\}

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).

\checkgputcode:nnn { begindocument } { . }
{ \IfPackageLoadedTF { colortbl }
\{ \ct@arc@ \}
\}

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

\checkgputcode:nnn { begindocument } { . }
{ \IfPackageLoadedTF { colortbl }
\{ \}
\}

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.

\checkgputcode:nnn { begindocument } { . }
{ \CT@arc@ \}

\arrayrulecolor \#1 # { \CT@arc@ \#1 }

We have to redefine \texttt{\cline} for several reasons. The command \texttt{\@@_cline} will be linked to \texttt{\cline} in the beginning of \texttt{NiceArrayWithDelims}. The following commands must \textit{not} be protected.

\begin{verbatim}
\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 } \}
\CS@arc@ \leaders \hrule \hfill
\skip_horizontal:N \c_zero_dim
\noalign { \skip_vertical:N -\arrayrulewidth }
\cr
\end{verbatim}

The following \texttt{\skip_horizontal:N \c_zero_dim} is to prevent a potential \texttt{\unskip} to delete the \texttt{\leaders}.

\begin{verbatim}
\everycr \texttt{\c_zero_dim}
\end{verbatim}

Our \texttt{\everycr} has been modified. In particular, the creation of the row node is in the \texttt{\everycr} (maybe we should put it with the incrementation of \texttt{\C@iRow}). Since the following \texttt{\cr} correspond to a “false row”, we have to nullify \texttt{\everycr}.

\begin{verbatim}
\everycr { }
\cr
\noalign { \skip_vertical:N -\arrayrulewidth }
\end{verbatim}

The following version of \texttt{\cline} spreads the array of a quantity equal to \texttt{\arrayrulewidth} as does \texttt{\hline}. It will be loaded excepted if the key \texttt{standard-cline} has been used.

\begin{verbatim}
\cs_set:Npn \@@_cline
\end{verbatim}

We have to act in a fully expandable way since there may be \texttt{\noalign} (in the \texttt{\multispan}) to detect. That’s why we use \texttt{\@@_cline_i:en}.

\begin{verbatim}
{ \@@_cline_i:en \l_@@_first_col_int }
\end{verbatim}

The command \texttt{\cline_i:nn} has two arguments. The first is the number of the current column (it \textit{must} be used in that column). The second is a standard argument of \texttt{\cline} of the form \texttt{i-j} or the form \texttt{i}.

\footnote{See question 99041 on TeX StackExchange.}
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).

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

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

The following command must not be protected since it will be used to write instructions in the (internal) \CodeBefore.
The following command must be protected because of its use of the command \color.

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

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

5 Parameters

The following counter will count the environments \texttt{\{NiceArray\}}. The value of this counter will be used to prefix the names of the Tikz nodes created in the array.

\verb|\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).

\verb|\cs_new:Npn \@@_env: { nm - \int_use:N \g_@@_env_int }|

The command \texttt{\NiceMatrixLastEnv} is not used by the package \texttt{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.

\verb|\NewExpandableDocumentCommand \NiceMatrixLastEnv { }|
\verb|{ \int_use:N \g_@@_env_int }|

The following command is only a syntaxic shortcut. The \texttt{q} in \texttt{qpoint} means \textit{quick}.

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

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

\verb|\bool_new:N \l_@@_tabular_bool|
\verb|\g_@@_delims_bool will be true for the environments with delimiters (ex. : \{pNiceMatrix\}, \{pNiceArray\}, \pAutoNiceMatrix, etc.).|
\verb|\bool_new:N \g_@@_delims_bool|
\verb|\bool_set_true:N \g_@@_delims_bool|

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

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

\verb|\bool_new:N \l_@@_preamble_bool|
\verb|\bool_set_true:N \l_@@_preamble_bool|
We need a special treatment for \{NiceMatrix\} when \texttt{vlines} is not used, in order to retrieve \texttt{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 \texttt{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 \texttt{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 \texttt{columns-width} is used with the special value \texttt{auto}, the boolean \l_@@_auto_columns_width_bool also will be raised).

\dim_new:N \l_@@_columns_width_dim
\dim_set:Nn \l_@@_columns_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 \texttt{@@_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 token list will contain the type of horizontal alignment of the current cell as provided by the corresponding column. The possible values are \texttt{r, l, c} and \texttt{j}. For example, a column \texttt{p[l]{3cm}} will provide the value \texttt{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 \texttt{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 \texttt{width} (which may be fixed in \texttt{NiceMatrixOptions} but also in an environment \{NiceTabular\}).

\dim_new:N \l_@@_width_dim
The sequence $\texttt{g@@names_seq}$ will be the list of all the names of environments used (via the option name) in the document: two environments must not have the same name. However, it’s possible to use the option allow-duplicate-names.

\seq_new:N \g@@names_seq

We want to know whether we are in an environment of \nicematrix because we will raise an error if the user tries to use nested environments.

\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

If the user uses \texttt{\{NiceTabular\*\}}, the width of the tabular (in the first argument of the environment \texttt{\{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 \texttt{color} in a command of rule such as \texttt{\Hline} (or the specifier “\texttt{|}” in the preamble of an environment).

\tl_new:N \l@@rule_color_tl

The following boolean will be raised when the command \texttt{\rotate} is used.

\bool_new:N \g@@rotate_bool

The following boolean will be raise then the command \texttt{\rotate} is used with the key \texttt{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 \texttt{X} thanks to that flag.

\bool_new:N \l@@X_bool

\bool_new:N \g@@caption_finished_bool

We will write in \texttt{\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 \texttt{\tl_gset:cn \{ c@@ \int_use:N \g@@env_int_tl \}}).

\tl_new:N \g@@aux_tl

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

\bool_new:N \g@@aux_found_bool

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

\seq_new:N \g@@size_seq

\tl_new:N \g@@left_delim_tl
\tl_new:N \g@@right_delim_tl

The token list \texttt{\g@@user_preamble_tl} will contain the preamble provided by the the final user of \nicematrix (eg the preamble of an environment \texttt{\{NiceTabular\}}).
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
\tl_new:N \g@columnstype_tl
\str_set:Nn \l@columnstype_tl { c }
\tl_new:N \l@iddots_down_tl
\tl_new:N \l@iddots_up_tl
\tl_new:N \l@iddots_middle_tl
\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 analyse 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
\tl_new:N \g@com_or_env_str
\tl_gset:Nn \g@com_or_env_str { environment }
\bool_new:N \l@bold_row_style_bool
The following command will be able to reconstruct the full name of the current command or environment (despite its name which contains env). This command must not be protected since it will be used in error messages and we have to use \str_if_eq:onTF and not \tl_if_eq:NnTF because we need to be fully expandable).
\cs_new:Npn \@full_name_env: { 
  \str_if_eq:onTF \g@com_or_env_str { command } {
    \str_if_eq:onTF \g@name_env_str { \space \c@backslash_str \g@name_env_str }
  { \str_if_eq:onTF \g@name_env_str { \space \g@name_env_str } 
    \str_if_eq:onTF \g@name_env_str { \space \g@name_env_str } 
  } 
}
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 splitted 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 splitted 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 content a `&` 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. The weight of a X-column is given as an optional argument between square brackets. The default value, of course, is 1.

\int_new:N \g_@@_total_X_weight_int

If there is at least one X-column in the preamble of the array, the following flag will be raised via the `aux` file. The length `\l_@@_X_columns_dim` will be the width of X-columns of weight 1 (the width of a column of weight $n$ will be that dimension multiplied by $n$). That value is computed after the construction of the array during the first compilation in order to be used in the following run.

\bool_new:N \l_@@_X_columns_aux_bool
\dim_new:N \l_@@_X_columns_dim
This boolean will be used only to detect in an expandable way whether we are at the beginning of the (potential) column zero, in order to raise an error if \Hdotsfor is used in that column.

\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 \cel 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 \cel 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

\l_@@_code_before_tl may contain two types of informations:

- A code-before written in the aux file by a previous run. When the aux file is read, this code-before is stored in \g_@@_code_before_i_tl (where i is the number of the environment) and, at the beginning of the environment, it will be put in \l_@@_code_before_tl.

- The final user can explicitly add material in \l_@@_code_before_tl by using the key code-before or the keyword \CodeBefore (with the keyword \Body).

\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

The \L3 programming layer provides scratch dimensions \l_tmpa_dim and \l_tmpb_dim. We creates two more in the same spirit.

\dim_new:N \l_@@_tmpc_dim
\dim_new:N \l_@@_tmpd_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.

418 \seq_new:N \g_@@_blocks_seq

We also manage a sequence of the positions of the blocks. In that sequence, each block is represented by only five components: \( \{imin\}\{jmin\}\{imax\}\{jmax\}\{name\} \). A block with the key hvlines won’t appear in that sequence (otherwise, the lines in that block would not be drawn!).

419 \seq_new:N \g_@@_pos_of_blocks_seq

In fact, this sequence will also contain the positions of the cells with a \diagbox. The sequence \g_@@_pos_of_blocks_seq will be used when we will draw the rules (which respect the blocks).

We will also manage a sequence for the positions of the dotted lines. These dotted lines are created in the array by \Cdots, \Vdots, \Ddots, etc. However, their positions, that is to say, their extremities, will be determined only after the construction of the array. In this sequence, each item contains five components: \( \{imin\}\{jmin\}\{imax\}\{jmax\}\{name\} \).

420 \seq_new:N \g_@@_pos_of_xdots_seq

The sequence \g_@@_pos_of_xdots_seq will be used when we will draw the rules required by the key hvlines (these rules won’t be drawn within the virtual blocks corresponding to the dotted lines).

The final user may decide to “stroke” a block (using, for example, the key \texttt{draw=red!15} when using the command \Block). In that case, the rules specified, for instance, by hvlines must not be drawn around the block. That’s why we keep the information of all that stroken blocks in the following sequence.

421 \seq_new:N \g_@@_pos_of_stroken_blocks_seq

If the user has used the key \texttt{corners}, all the cells which are in an (empty) corner will be stored in the following sequence.

422 \seq_new:N \l_@@_corners_cells_seq

The list of the names of the potential \SubMatrix in the \CodeAfter of an environment. Unfortunately, that list has to be global (we have to use it inside the group for the options of a given \SubMatrix).

423 \seq_new:N \g_@@_submatrix_names_seq

The following flag will be raised if the key \texttt{width} is used in an environment \texttt{NiceTabular} (not in a command \texttt{NiceMatrixOptions}). You use it to raise an error when this key is used while no column \texttt{X} is used.

424 \bool_new:N \l_@@_width_used_bool

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

425 \seq_new:N \g_@@_multicolumn_cells_seq
426 \seq_new:N \g_@@_multicolumn_sizes_seq

The following counters will be used when searching the extremities of a dotted line (we need these counters because of the potential “open” lines in the \SubMatrix—the \SubMatrix in the \texttt{code-before}).

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

15
The following counters will be used when drawing the rules.
\[\int_{\text{new:N}} \backslash \_\_\_\_\text{start}\_\text{int}\]
\[\int_{\text{set_eq:NN}} \backslash \_\_\_\_\text{start}\_\text{int} \ \text{c}\_\text{one}\_\text{int}\]
\[\int_{\text{new:N}} \backslash \_\_\_\_\text{end}\_\text{int}\]
\[\int_{\text{new:N}} \backslash \_\_\_\_\text{local}\_\text{start}\_\text{int}\]
\[\int_{\text{new:N}} \backslash \_\_\_\_\text{local}\_\text{local}\_\text{end}\_\text{int}\]

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.

\[\seq_{\text{new:N}} \backslash \_\_\_\_\text{submatrix_seq}\]

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).
\[\int_{\text{new:N}} \backslash \_\_\_\_\text{static}\_\text{num}\_\text{of}\_\text{col}\_\text{int}\]

The following parameters correspond to the keys \texttt{fill}, \texttt{opacity}, \texttt{draw}, \texttt{tikz}, \texttt{borders}, and \texttt{rounded-corners} of the command \Block.
\[\tl_{\text{new:N}} \backslash \_\_\_\_\text{fill}\_\text{tl}\]
\[\tl_{\text{new:N}} \backslash \_\_\_\_\text{opacity}\_\text{tl}\]
\[\tl_{\text{new:N}} \backslash \_\_\_\_\text{draw}\_\text{tl}\]
\[\seq_{\text{new:N}} \backslash \_\_\_\_\text{tikz}\_\text{seq}\]
\[\clist_{\text{new:N}} \backslash \_\_\_\_\text{borders}\_\text{clist}\]
\[\dim_{\text{new:N}} \backslash \_\_\_\_\text{rounded}\_\text{corners}\_\text{dim}\]

The last parameter has no direct link with the [empty] corners of the array (which are computed and taken into account by \nicematrix when the key \texttt{corners} is used).

The following dimension corresponds to the key \texttt{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.
\[\dim_{\text{new:N}} \backslash \_\_\_\_\text{tab}\_\text{rounded}\_\text{corners}\_\text{dim}\]

The following token list correspond to the key \texttt{color} of the command \Block and also the key \texttt{color} of the command \RowStyle.
\[\tl_{\text{new:N}} \backslash \_\_\_\_\text{color}\_\text{tl}\]

In the key \texttt{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 \texttt{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.
\[\dim_{\text{new:N}} \backslash \_\_\_\_\text{offset}\_\text{dim}\]

Here is the dimension for the width of the rule when a block (created by \Block) is stroked.
\[\dim_{\text{new:N}} \backslash \_\_\_\_\text{line}\_\text{width}\_\text{dim}\]

The parameters of the horizontal position of the label of a block. If the user uses the key \texttt{c} or \texttt{C}, the value is \texttt{c}. If the user uses the key \texttt{1} or \texttt{L}, the value is \texttt{1}. If the user uses the key \texttt{r} or \texttt{R}, the value is \texttt{r}. If the user has used a capital letter, the boolean \_\_\_\_\text{hpos}\_\text{of}\_\text{block}\_\text{cap}\_\text{bool} will be raised (in the second pass of the analyze of the keys of the command \Block).
\[\str_{\text{new:N}} \backslash \_\_\_\_\text{hpos}\_\text{block}\_\text{str}\]
\[\str_{\text{set:NN}} \backslash \_\_\_\_\text{hpos}\_\text{block}\_\text{str} \ \{ \text{c} \} \]
\[\bool_{\text{new:N}} \backslash \_\_\_\_\text{hpos}\_\text{of}\_\text{block}\_\text{cap}\_\text{bool}\]
\[\bool_{\text{new:N}} \backslash \_\_\_\_\text{p}\_\text{block}\_\text{bool}\]

If the final user has used the special color \texttt{"nocolor"}, the following flag will be raised.
\[\bool_{\text{new:N}} \backslash \_\_\_\_\text{nocolor}\_\text{used}\_\text{bool}\]
For the vertical position, the possible values are c, t and b.
\str_new:N \l_@@_vpos_block_str
\str_set:Nn \l_@@_vpos_block_str { c }

Used when the key \texttt{draw-first} is used for \textbackslash Ddots or \textbackslash 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 \texttt{--} 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 \texttt{rounded-corners} (for a \{tabular\}).
\bool_new:N \l_@@_hvlines_bool

The following flag will be used by (for instance) \texttt{\@@_vline_i}
. 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 \texttt{caption}).
\bool_new:N \l_@@_in_caption_bool

Variables for the exterior rows and columns

The keys for the exterior rows and columns are \texttt{first-row}, \texttt{first-col}, \texttt{last-row} and \texttt{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 \texttt{first-row} is used, the value will be 0.
\int_new:N \l_@@_first_row_int
\int_set:Nn \l_@@_first_row_int 1

- **First column**
  The integer \l_@@_first_col_int is the number of the first column of the array. The default value is 1, but, if the option \texttt{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 \texttt{\l_@@\_last\_row\_int} is the number of the potential “last row”, as specified by the key \texttt{last-row}. A value of \texttt{-2} means that there is no “last row”. A value of \texttt{-1} means that there is a “last row” but we don’t know the number of that row (the key \texttt{last-row} has been used without value and the actual value has not still been read in the aux file).

\begin{verbatim}
473 \int_new:N \l_@@_last_row_int
474 \int_set:Nn \l_@@_last_row_int { -2 }
\end{verbatim}

If, in an environment like \texttt{\{pNiceArray\}}, the option \texttt{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”.

\begin{verbatim}
475 \bool_new:N \l_@@_last_row_without_value_bool
\end{verbatim}

Idem for \texttt{\l_@@\_last\_col\_without\_value\_bool}

\begin{verbatim}
476 \bool_new:N \l_@@_last_col_without_value_bool
\end{verbatim}

• Last column

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

\begin{verbatim}
477 \int_new:N \l_@@_last_col_int
478 \int_set:Nn \l_@@_last_col_int { -2 }
\end{verbatim}

However, we have also a boolean. Consider the following code:

\begin{verbatim}
\verb|\begin{pNiceArray}{cc}[last-col}\verb| 1 & 2 \verb|\end{pNiceArray}|
\end{verbatim}

In such a code, the “last column” specified by the key \texttt{last-col} is not used. We want to be able to detect such a situation and we create a boolean for that job.

\begin{verbatim}
479 \bool_new:N \g_@@_last_col_found_bool
\end{verbatim}

This boolean is set to \texttt{false} at the end of \texttt{\@@\_pre\_array\_ii}.

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

\begin{verbatim}
480 \bool_new:N \l_@@_in_last_col_bool
\end{verbatim}

Some utilities

\begin{verbatim}
481 \cs_set_protected:Npn \@@\_cut\_on\_hyphen:w #1-#2\q_stop
482 { \cs_set_nopar:Npn \l_tmpa_tl { #1 } \cs_set_nopar:Npn \l_tmpb_tl { #2 } }
\end{verbatim}

\footnote{We can’t use \texttt{\l_@@\_last\_row\_int} for this usage because, if \texttt{nicematrix} has read its value from the aux file, the value of the counter won’t be \texttt{-1} any longer.}
The following takes as argument the name of a \texttt{clist} and which should be a list of intervals of integers. It \textit{expands} that list, that is to say, it replaces (by a sort of \texttt{mapcan} or \texttt{flat_map}) the interval by the explicit list of the integers.

\begin{verbatim}
\cs_new_protected:Npn \@@_expand_clist:N #1  
{  
\clist_if_in:NVF #1 \c_@@_all_tl  
{  
\clist_clear:N \l_tmpa_clist  
\clist_map_inline:Nn #1  
{  
\tl_if_in:nnTF { ##1 } { - }  
{  \@@_cut_on_hyphen:w ##1 \q_stop }  
{  \cs_set_nopar:Npn \l_tmpa_tl { ##1 }  
\cs_set_nopar:Npn \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  
}
\end{verbatim}

The following internal parameters are for:

- \texttt{\Ldots} with both extremities open (and hence also \texttt{\Hdots} for in an exterior row;
- \texttt{\Vdots} with both extremities open (and hence also \texttt{\Vdots} for in an exterior column;
- when the special character “:” is used in order to put the label of a so-called “dotted line” \textit{on the line}, a margin of \texttt{\c_@@_innersep_middle_dim} will be added around the label.

\begin{verbatim}
\hook_gput_code:nnn { begindocument } { . }  
{  
\dim_const:Nn \c_@@_shift_Ldots_last_row_dim { 0.5 em }  
\dim_const:Nn \c_@@_shift_exterior_Vdots_dim { 0.6 em }  
\dim_const:Nn \c_@@_innersep_middle_dim { 0.17 em }  
}
\end{verbatim}

6 The command \texttt{\tabularnote}

Of course, it’s possible to use \texttt{\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 \texttt{\caption} in a floating environment. Of course, a command \texttt{\tabularnote} in that \texttt{\caption} makes sens only if the \texttt{\caption} is \textit{before} the \{tabular\}.
- It’s also possible to use \texttt{\tabularnote} in the value of the key \texttt{\caption} of the \{NiceTabular\} when the key \texttt{\caption-above} is in force. However, in that case, one must remind that the caption is composed \textit{after} the composition of the box which contains the main tabular (that’s mandatory since that caption must be wrapped with a line width equal to the width of the tabular). However, we want the labels of the successive tabular notes in the logical order. That’s why:
The number of tabular notes present in the caption will be written on the aux file and available in \g_@@_notes_caption_int.\(^3\)

During the composition of the main tabular, the tabular notes will be numbered from \g_@@_notes_caption_int+1 and the notes will be stored in \g_@@_notes_seq. Each component of \g_@@_notes_seq will be a kind of couple of the form : \{label\}{text of the tabularnote}. The first component is the optional argument (between square brackets) of the command \tabularnote (if the optional argument is not used, the value will be the special marker expressed by \c_novalue_tl).

During the composition of the caption (value of \l_@@_caption_tl), the tabular notes will be numbered from 1 to \g_@@_notes_caption_int and the notes themselves will be stored in \g_@@_notes_in_caption_seq. The structure of the components of that sequence will be the same as for \g_@@_notes_seq.

After the composition of the main tabular and after the composition of the caption, the sequences \g_@@_notes_in_caption_seq and \g_@@_notes_seq will be merged (in that order) and the notes will be composed.

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

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.

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.

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

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

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

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 \g_@@_notes_style:n.

\(^3\)More precisely, it’s the number of tabular notes which do not use the optional argument of \tabularnote.
The tabular notes will be available for the final user only when \texttt{enumitem} is loaded. Indeed, the tabular notes will be composed at the end of the array with a list customized by \texttt{enumitem} (a list \texttt{tabularnotes} in the general case and a list \texttt{tabularnotes*} if the key \texttt{para} is in force). However, we can test whether \texttt{enumitem} has been loaded only at the beginning of the document (we want to allow the user to load \texttt{enumitem} after \texttt{nicematrix}).

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

The type of list \texttt{tabularnotes} will be used to format the tabular notes at the end of the array in the general case and \texttt{tabularnotes*} will be used if the key \texttt{para} is in force.

\begin{verbatim}
\newlist { tabularnotes } { enumerate } { 1 }
\setlist [ tabularnotes ]
\{ topsep = 0pt ,
   noitemsep ,
   leftmargin = * ,
   align = left ,
   labelsep = 0pt ,
   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 } } ,
\}
\end{verbatim}

One must remind that we have allowed a \texttt{tabular} in the caption and that caption may also be found in the list of tables (\texttt{listoftables}). We want the command \texttt{tabularnote} be no-op during the composition of that list. That’s why we program \texttt{tabularnote} to be no-op excepted in a floating environment or in an environment of \texttt{nicematrix}.

\begin{verbatim}
\NewDocumentCommand \tabularnote { o m }
{ \bool_lazy_or:nnT \{ \cs_if_exist_p:N \@captype \} \_\@@_in_env_bool
  { \bool_lazy_and:nnTF \{ ! \_\@@_tabular_bool \} \_\@@_in_env_bool
    { \_\@@_error:n { tabularnote-forbidden } }
    { \_\@@_if:NTF \_\@@_in_caption_bool \_\@@_tabularnote_caption:nn
      \_\@@_tabularnote:nn
      { #1 } { #2 }
    }
  }
}
\end{verbatim}
For the version in normal conditions, that is to say not in the `caption`. 

#1 is the optional argument of \texttt{\tabularnote} (maybe equal to the special marker expressed by \texttt{\c_novalュe_tl}) and #2 is the mandatory argument of \texttt{\tabularnote}.

You have to see whether the argument of \texttt{\tabularnote} has yet been used as argument of another \texttt{\tabularnote} in the same tabular. In that case, there will be only one note (for both commands \texttt{\tabularnote}) at the end of the tabular. We search the argument of our command \texttt{\tabularnote} in \texttt{\g__00_notes_seq}. The position in the sequence will be stored in \texttt{\l_tmpa_int} (0 if the text is not in the sequence yet).

\cs_new_protected:Npn \@@_tabularnote:nn #1 #2
\begin{itemize}
\item You have to see whether the argument of \texttt{\tabularnote} has yet been used as argument of another \texttt{\tabularnote} in the same tabular. In that case, there will be only one note (for both commands \texttt{\tabularnote}) at the end of the tabular. We search the argument of our command \texttt{\tabularnote} in \texttt{\g__00_notes_seq}. The position in the sequence will be stored in \texttt{\l_tmpa_int} (0 if the text is not in the sequence yet).
\end{itemize}

\begin{itemize}
\item \texttt{\int_zero:N \l_tmpa_int}
\item \texttt{\bool_if:NT \l__00_notes_detect_duplicataes_bool}
\end{itemize}

We recall that each component of \texttt{\g__00_notes_seq} is a kind of couple of the form

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

If the user have used \texttt{\tabularnote} without the optional argument, the \texttt{label} will be the special marker expressed by \texttt{\c_novalュe_tl}.

When we will go through the sequence \texttt{\g__00_notes_seq}, we will count in \texttt{\l_tmpb_int} the notes without explicit label in order to have the “current” value of the counter \texttt{\c@tabularnote}.

\begin{itemize}
\item \texttt{\int_zero:N \l_tmpb_int}
\item \texttt{\seq_map_indexed_inline:Nn \g__00_notes_seq}
\end{itemize}

\begin{itemize}
\item \texttt{\@@_test_first_novalue:nnn ##2 { \int_incr:N \l_tmpb_int }}
\item \texttt{\tl_if_eq:nnT { \{ #1 \} } { #2 } { #2 }}
\item \texttt{\tl_if_novalue:nTF { #1 }}
\item \texttt{\tl_if_novalue:nT \{ #1 \}}
\item \texttt{\seq_map_break:}
\end{itemize}

\begin{itemize}
\item \texttt{\int_if_zero:nF \l_tmpa_int}
\item \texttt{\int_add:Nn \l_tmpa_int \g__00Notes_caption_int}
\end{itemize}

\begin{itemize}
\item \texttt{\int_if_zero:nT \l_tmpa_int}
\item \texttt{\seq_gput_right:Nn \g__00Notes_seq \{ \{ #1 \} } { #2 }
\item \texttt{\int_if_novalue:nTF \{ #1 \}} \texttt{\int_gincr:N \c@tabularnote}
\end{itemize}

\begin{itemize}
\item \texttt{\seq_put_right:Nx \l__00Notes_labels_seq}
\item \texttt{\int_if_novalue:nTF \{ #1 \}}
\item \texttt{\seq__0Notes_format:n}
\item \texttt{\int_eval:n}
\item \texttt{\int_if_zero:nTF \l_tmpa_int}
\item \texttt{\c@tabularnote}
\item \texttt{\l_tmpa_int}
\end{itemize}

\begin{itemize}
\item \texttt{\peek_meaning:NF \tabularnote}
\end{itemize}

If the following token is not a \texttt{\tabularnote}, we have finished the sequence of successive commands \texttt{\tabularnote} and we have to format the labels of these tabular notes (in the array). We compose
those labels in a box $\l@l_{\text{tmpa_box}}$ 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 $\l@l_{\_\_@\_hpos\_cell\_tl}$ is equal to $c$ or $r$.

\box_set:Nn \l@l_{\text{tmpa_box}}
\{
We remind that it is the command $\@@@notes_label_in_tabular:n$ that will put the labels in a $\text{textsuperscript}$.

\@@@notes_label_in_tabular:n
\{
\seq_use:Nnnn \l@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@l_{\_@\_@int_tempa_int} \c@tabularnote
\refstepcounter{tabularnote}
\int_compare:nNnT \l@l_{\_@\_@int_tempa_int} = \c@tabularnote
\{
\int_gincr:N \c@tabularnote
\}
\seq_clear:N \l@l_{\_@\_@notes_labels_seq}
\bool_lazy_or:nnTF
\{
\tl_if_eq_p:NN \l@l_{\_@\_@hpos\_cell\_tl} \c@\_@\_c\_tl
\}
\{
\tl_if_eq_p:NN \l@l_{\_@\_@hpos\_cell\_tl} \c@\_@\_r\_tl
\}
\hbox_overlap_right:n \{ \box_use:N \l@l_{\text{tmpa_box}} \}

If the command $\text{tabularnote}$ is used exactly at the end of the cell, the $\text{unskip}$ (inserted by $\text{array}$?) 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@l_{\text{tmpa_box}} \}
\}
\{
\box_use:N \l@l_{\text{tmpa_box}} \}
\}

Now the version when the command is used in the key $\text{caption}$. The main difficulty is that the argument of the command $\text{caption}$ is composed several times. In order to know the number of commands $\text{tabularnote}$ 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 $\text{caption}$.

\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
\}
\seq_if_in:NnF \g@_@notes\_in\_caption\_seq { { #1 } { #2 } }
\{
\@@@error:n \{ Identical-notes-in-caption \}
\}
\}
\\}
\}

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 $\text{tabularnote}$ of the second composition.

\seq_if_in:NnF \g@_@notes\_in\_caption\_seq { { #1 } { #2 } }
\{
\}

23
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 { \detokenize{#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:Nx \l@@notes_labels_seq { \@@_notes_format:n { \int_use:N \c@tabularnote } { #1 } }
\peek_meaning:NF \tabularnote {
\@@_notes_label_in_tabular:n { \seq_use:Nnnn \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 } }

7 Command for creation of rectangle nodes

The following command should be used in a \{pgfpicture\}. It creates a rectangle (empty but with a name).

#1 is the name of the node which will be created; #2 and #3 are the coordinates of one of the corner of the rectangle; #4 and #5 are the coordinates of the opposite corner.

\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:nnnn: 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 } { }
\end { pgfscope }

8 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 parameter 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 commands \cellcolor and \rowcolor are available for the user in the cells of the tabular (the user may use the commands provided by \colortbl). However, if the key color-inside is used, these commands are available.

\bool_new:N \l_@@_color_inside_bool

By default, the behaviour of \cline is changed in the environments of \nicematrix: a \cline spreads the array by an amount equal to \arrayrulewidth. It’s possible to disable this feature with the key \l_@@_standard_line_bool.

\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
\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 \texttt{true}.

\begin{verbatim}
\bool_new:N \l_@@_parallelize_diags_bool
\bool_set_true:N \l_@@_parallelize_diags_bool
\end{verbatim}

The following parameter correspond to the key \texttt{corners}. The elements of that \texttt{clist} must be within NW, SW, NE and SE.

\begin{verbatim}
\clist_new:N \l_@@_corners_clist
\end{verbatim}

\begin{verbatim}
\dim_new:N \l_@@_notes_above_space_dim
\hook_gput_code:nnn { begindocument } { . }
{ \dim_set:Nn \l_@@_notes_above_space_dim { 1 mm } }
\end{verbatim}

We use a hook only by security in case \texttt{revtex4-1} is used (even though it is obsolete).

The flag \l_@@_nullify_dots_bool corresponds to the option \texttt{nullify-dots}. When the flag is down, the instructions like \texttt{\vdots} are inserted within a \texttt{\hphantom} (and so the constructed matrix has exactly the same size as a matrix constructed with the classical \{\texttt{matrix}\} and \texttt{\ldots}, \texttt{\vdots}, etc.).

\begin{verbatim}
\bool_new:N \l_@@_nullify_dots_bool
\end{verbatim}

When the key \texttt{respect-arraystretch} is used, the following command will be nullified.

\begin{verbatim}
\cs_new_protected:Npn \@@_reset_arraystretch:
{ \cs_set_nopar:Npn \arraystretch { 1 } }
\end{verbatim}

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).

\begin{verbatim}
\bool_new:N \l_@@_auto_columns_width_bool
\end{verbatim}

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

\begin{verbatim}
\bool_new:N \g_@@_recreate_cell_nodes_bool
\end{verbatim}

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.

\begin{verbatim}
\str_new:N \l_@@_name_str
\end{verbatim}

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”.

\begin{verbatim}
\bool_new:N \l_@@_medium_nodes_bool
\bool_new:N \l_@@_large_nodes_bool
\end{verbatim}

The boolean \l_@@_except_borders_bool will be raised when the key \texttt{hvlines-except-borders} will be used (but that key has also other effects).

\begin{verbatim}
\bool_new:N \l_@@_except_borders_bool
\end{verbatim}

The dimension \l_@@_left_margin_dim correspond to the option \texttt{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 (\texttt{\hdottedline}).

\begin{verbatim}
\dim_new:N \l_@@_left_margin_dim
\dim_new:N \l_@@_right_margin_dim
\end{verbatim}

The dimensions \l_@@_extra_left_margin_dim and \l_@@_extra_right_margin_dim correspond to the options \texttt{extra-left-margin} and \texttt{extra-right-margin}.

\begin{verbatim}
\dim_new:N \l_@@_extra_left_margin_dim
\dim_new:N \l_@@_extra_right_margin_dim
\end{verbatim}
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.

\begin{verbatim}
\tl_new:N \l_@@_end_of_row_tl
\tl_set:Nn \l_@@_end_of_row_tl { ; }
\end{verbatim}

The following parameter is for the color the dotted lines drawn by \Cdots, \Ldots, \Vdots, \Ddots, \Iddots and \Hdots for but not the dotted lines drawn by \hdottedline and ":".

\begin{verbatim}
\tl_new:N \l_@@_xdots_color_tl
\end{verbatim}

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

\begin{verbatim}
\tl_new:N \l_@@_delimiters_color_tl
\end{verbatim}

Sometimes, we want to have several arrays vertically juxtaposed in order to have an alignment of the columns of these arrays. To achieve this goal, one may wish to use the same width for all the columns (for example with the option columns-width or the option auto-columns-width of the environment \{NiceMatrixBlock\}). However, even if we use the same type of delimiters, the width of the delimiters may be different from an array to another because the width of the delimiter is a function 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.

\begin{verbatim}
\bool_new:N \l_@@_delimiters_max_width_bool
\keys_define:nn { NiceMatrix / xdots } { 
  shorten-start .code:n = \hook_gput_code:nnn { begindocument } { . } { \dim_set:Nn \l_@@_xdots_shorten_start_dim { #1 } },
  shorten-end .code:n = \hook_gput_code:nnn { begindocument } { . } { \dim_set:Nn \l_@@_xdots_shorten_end_dim { #1 } },
  shorten-start .value_required:n = true ,
  shorten-end .value_required:n = true ,
  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 ,
  horizontal-labels .bool_set:N = \l_@@_xdots_h_labels_bool ,
  horizontal-labels .default:n = true ,
  line-style .code:n = \hook_gput_code:nnn { begindocument } { . } { \dim_set:Nn \l_@@_xdots_line_style_tl { #1 } },
  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 ,
}\end{verbatim}
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 ^{...}.

\begin{verbatim}
\newcommand{\@_xdots_down_tl}{#1}
\newcommand{\@_xdots_up_tl}{#1}
\newcommand{\@_xdots_middle_tl}{#1}
\end{verbatim}

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

\begin{verbatim}
\NewDocumentCommand{\draw-first}{o}{\IfNoValueTF{\Arg{1}}{\prg_do_nothing:}{\@_error:n {Unknown key for xdots}}}
\end{verbatim}

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

\begin{verbatim}
\keys_define:nn { NiceMatrix / Global }
\end{verbatim}

\begin{verbatim}
\keys_define:nn { NiceMatrix / rules }
\end{verbatim}
We write directly a command for the automata which reads the preamble provided by the final user.

\begin{verbatim}
\cs_set_eq:cN { @@ _ #1 } \@@_make_preamble_vlism:n
\end{verbatim}

With the option `renew-dots`, the command `\cdots`, `\ldots`, `\vdots`, `\ddots`, etc. are redefined and behave like the commands `\Cdots`, `\Ldots`, `\Vdots`, `\Ddots`, etc.
We define a set of keys used by the environments of \texttt{nicematrix} (but not by the command \texttt{\textbackslash NiceMatrixOptions}).

\begin{verbatim}
\keys_define:nn { NiceMatrix / Env }
{
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 ,
\}
\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 ,
}\end{verbatim}

The options \texttt{c}, \texttt{t} and \texttt{b} of the environment \texttt{\{NiceArray\}} have the same meaning as the option of the classical environment \texttt{\{array\}}.

\begin{verbatim}
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 =
{\tl_if_eq:nnTF { #1 } { auto }
{\dim_set:Nn \l_@@_columns_width_dim { #1 } } ,
columns-width .value_required:n = true ,
name .code:n =
\legacy_if:nF { measuring@ }
{\str_set:Nx \l_tmpa_str { #1 }
\seq_if_in:NVTF \g_@@_names_seq \l_tmpa_str
{ \@@_error:nn { Duplicate~name } { #1 } }
{ \seq_gput_left:NV \g_@@_names_seq \l_tmpa_str }
\str_set_eq:NN \l_@@_name_str \l_tmpa_str }
},
name .value_required:n = true ,
code-after .tl_gset:N = \g_nicematrix_code_after_tl ,
code-after .value_required:n = true ,
color-inside .code:n =
{\bool_set_true:N \l_@@_color_inside_bool
\bool_set_true:N \l_@@_code_before_bool ,
color-inside .value_forbidden:n = true ,
colorbl-like .meta:n = color-inside }
}\end{verbatim}

We test whether we are in the measuring phase of an environment of \texttt{amsmath} (always loaded by \texttt{nicematrix}) because we want to avoid a fallacious message of duplicate name in this case.

\begin{verbatim}
\legacy_if:nF { measuring@ }
{\str_set:Nx \l_tmpa_str { #1 }
\seq_if_in:NVTF \g_@@_names_seq \l_tmpa_str
{ \@@_error:nn { Duplicate~name } { #1 } }
{ \seq_gput_left:NV \g_@@_names_seq \l_tmpa_str }
\str_set_eq:NN \l_@@_name_str \l_tmpa_str }
},
name .value_required:n = true ,
code-after .tl_gset:N = \g_nicematrix_code_after_tl ,
code-after .value_required:n = true ,
color-inside .code:n =
{\bool_set_true:N \l_@@_color_inside_bool
\bool_set_true:N \l_@@_code_before_bool ,
color-inside .value_forbidden:n = true ,
colorbl-like .meta:n = color-inside }
}\end{verbatim}
We begin the construction of the major sets of keys (used by the different user commands and environments).

\keys_define:nn { NiceMatrix / delimiters }
\keys_define:nn { NiceMatrix }
\keys_define:nn { NiceMatrixOptions .inherit:n = NiceMatrix / Global },
\keys_define:nn { NiceMatrixOptions / xdots .inherit:n = NiceMatrix / xdots },
\keys_define:nn { NiceMatrixOptions / rules .inherit:n = NiceMatrix / rules },
\keys_define:nn { NiceMatrixOptions / notes .inherit:n = NiceMatrix / notes },
\keys_define:nn { NiceMatrixOptions / sub-matrix .inherit:n = NiceMatrix / sub-matrix },
\keys_define:nn { CodeAfter / xdots .inherit:n = NiceMatrix / xdots },
\keys_define:nn { CodeAfter / sub-matrix .inherit:n = NiceMatrix / sub-matrix },
\keys_define:nn { CodeBefore / sub-matrix .inherit:n = NiceMatrix / sub-matrix },
\keys_define:nn { NiceMatrix .inherit:n = NiceMatrix / Global },
\keys_define:nn { NiceMatrix / Env },
\keys_define:nn { NiceMatrix / xdots .inherit:n = NiceMatrix / xdots },
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 = \tl_if_eq:nnTF { #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 distinct environments of \nicematrix (these names are global and not local to the current TeX scope). However, the option allow-duplicate-names disable this feature.

\allow-duplicate-names .code:n =
\@@_msg_redirect_name:nn { Duplicate-name } { none },
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 = \@@_error:n { 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 = \@@_error:n { 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 = \@@_error:n { 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 = \@@_error:n { 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 \textit{width} will be used if at least a column of type \textit{X} is used. If there is no column of type \textit{X}, an error will be raised.

  width .code:n = \dim_set:Nn \l_@@_width_dim { #1 }
  \bool_set_true:N \l_@@_width_used_bool ,
  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 = \@@_error:n { Unknown-key-for-NiceTabular }
}

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

\texttt{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 ,
}
9 Important code used by \texttt{\{NiceArrayWithDelims\}}

The pseudo-environment \texttt{\@@\_cell\_begin:w/\@@\_cell\_end:} will be used to format the cells of the array. In the code, the affectations are global because this pseudo-environment will be used in the cells of a \texttt{\halign} (via an environment \texttt{\{array\}).

\begin{verbatim}
\cs_new_protected:Npn \@@\_cell\_begin:w
\{
\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 \begin\ning \of \the \cell, \we \link \CodeAfter \to \a \command \which \do \be \gin\n with \texttt{\textbackslash} (\whereas \the \standard \version \of \CodeAfter \do\n not).
\cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
\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 \do \not \do \this \incrementation \in \the \everycr because \some \packages, \like \texttt{arydshln}, \create \special \rows \in \the \texttt{\halign} \that \we \do \not \want \to \take \into \account.
\int_compare:nNnT \c@iRow \= \c@\text{one\_int}
{ \int_compare:nNnT \c@jCol \> \c@\text{zero\_int}
{ \l_@@_code\_for\_first\_row\_tl
\xglobal \colorlet { nicematrix-first-row } { . } }
}
\The \content \of \the \cell \is \composed \in \the \box \l_@@_cell\_box. \The \texttt{\hbox_set\_end:} \corresponding \to \this \\texttt{\hbox_set:Nw} \is \in \the \texttt{\@@\_cell\_end:}.
\hbox_set:Nw \l_@@_cell\_box
\The \following \command \is \nullified \in \the \texttt{tabulars}.
\@@\_tuning\_not\_tabular\_begin:
\@@\_tuning\_first\_row:
\@@\_tuning\_last\_row:
\g_@@\_row\_style\_tl
\}
\The \following \command \will \be \nullified \unless \there \is \a \first \row.
\cs_new_protected:Npn \@@\_tuning\_first\_row:
{ \int_if_zero:nT \c@iRow
{ \int_compare:nNnT \c@jCol \> \c@\text{zero\_int}
{ \l_@@\_code\_for\_first\_row\_tl
\xglobal \colorlet { nicematrix-first-row } { . } }
}
}
\end{verbatim}
The following command will be nullified unless there is a last row and we know its value (ie: \l_@@_lat_row_int > 0).
\begin{verbatim}
\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 } { . } \}
\}
\end{verbatim}
A different value will be provided to the following command when the key small is in force.
\begin{verbatim}
\cs_set_eq:NN \@@_tuning_key_small: \prg_do_nothing:
\end{verbatim}
The following commands are nullified in the tabulars.
\begin{verbatim}
\cs_set_nopar:Npn \@@_tuning_not_tabular_begin:
\{ \c_math_toggle_token \}
\end{verbatim}
A special value is provided by the following controls sequence when the key small is in force.
\begin{verbatim}
\@@_tuning_key_small:
\end{verbatim}
\begin{verbatim}
\cs_set_eq:NN \@@_tuning_not_tabular_end: \c_math_toggle_token
\end{verbatim}
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.
\begin{verbatim}
\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 \l_@@_cell_box }
\dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \l_@@_cell_box }
\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
\end{verbatim}
Remark: If the key recreate-cell-nodes of the \CodeBefore is used, then we will add some lines to that command.

The following code is used in each cell of the array. It actualises quantities that, at the end of the array, will give informations about the vertical dimension of the two first rows and the two last rows. If the user uses the last-row, some lines of code will be dynamically added to this command.
\int_compare:nNnT \c@iRow = \c_one_int
{
    \dim_gset:Nn \g_@@_ht_row_one_dim
    \dim_max:nn \g_@@_ht_row_one_dim { \box_ht:N \l_@@_cell_box }
}
}
}

\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
        { \c_math_toggle_token
            \vcenter { \box_use:N \l_@@_cell_box }
            \c_math_toggle_token
        }
    }
    {
        \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
}
\cs_new_protected:Npn \@@_adjust_size_box:
{
    \dim_compare:nNnT \g_@@_blocks_wd_dim > \c_zero_dim
    {
        \box_set_wd:Nn \l_@@_cell_box
        { \dim_max:nn { \box_wd:N \l_@@_cell_box } \g_@@_blocks_wd_dim }
    }
    \dim_gzero:N \g_@@_blocks_wd_dim
}
\dim_compare:nNnT \g_@@_blocks_dp_dim > \c_zero_dim
{
    \box_set_dp:Nn \l_@@_cell_box
    { \dim_max:nn { \box_dp:N \l_@@_cell_box } \g_@@_blocks_dp_dim }
}
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.

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”).

The following computations are for the “first row” and the “last row”.

If the cell is empty, or may be considered as if, we must not create the PGF node, for two reasons:
- it’s a waste of time since such a node would be rather pointless;
- we test the existence of these nodes in order to determine whether a cell is empty when we search the extremities of a dotted line.

However, it’s very difficult to determine whether a cell is empty. Up to now we use the following 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.

The following command will be nullified in our redefinition of \ multicolumn.
The following variant of \texttt{\@\_cell\_end:} is only for the columns of type \texttt{w(s)}\ldots{} or \texttt{W(s)}\ldots{} (which use the horizontal alignment key \texttt{s} of \texttt{makebox}).

\begin{verbatim}
\cs_new_protected:Npn \@\_cell\_end\_for\_w\_s:\n{
  \@@\_math\_toggle:
  \hbox\_set\_end:
  \bool_if:FN \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:
}
\end{verbatim}

The following command creates the PGF name of the node with, of course, \texttt{\l_\@\_cell\_box} as the content.

\begin{verbatim}
\cs_new_protected:Npn \@\_node\_for\_cell:\n{\pgfpicture\pgfset\{
icematrix / cell-node /.style =
  \{
    inner\-sep = \c\_zero\_dim,
    minimum\-width = \c\_zero\_dim
  \}\pgfnode\{ rectangle \}{ base \}
  \pgfnodealias \{ \l_\@\_name\_str - \int\_use:N \c\@\_iRow - \int\_use:N \c\@\_jCol \}
  \str_if_empty:NF \l_\@\_name\_str
  \{
    \pgfnodealias
    \{ \l_\@\_name\_str - \int\_use:N \c\@\_iRow - \int\_use:N \c\@\_jCol \}
  \}
  \endpgfpicture}
\end{verbatim}

The following instruction \texttt{\set\_color} has been added on 2022/10/06. It’s necessary only with Xe-LaTeX and not with the other engines (we don’t know why).

\begin{verbatim}
\set\_color
\box\_use\_drop:N \l_\@\_cell\_box
\}
\endpgfpicture
\end{verbatim}

As its name says, the following command is a patch for the command \texttt{\@\_node\_for\_cell:}. This patch will be appended on the left of \texttt{\@\_node\_for\_the\_cell:} when the construction of the cell nodes (of the form \texttt{(i\textendash{}j)}) in the \texttt{\CodeBefore} is required.

\begin{verbatim}
\cs_new_protected:Npn \@\_patch\_node\_for\_cell:n #1
{
  \cs_new_protected:Npn \@\_patch\_node\_for\_cell:
    \{
      \set\_color
      \box\_use\_drop:N \l_\@\_cell\_box
    }
\end{verbatim}

40
I don’t know why the following adjustment is needed when the compilation is done with XeLaTeX or with the classical way latex, dvips, ps2pdf (or Adobe Distiller). However, it seems to work.

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

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).
\begin{verbatim}
\cs_new_protected:Npn \@@_array:
{\begin{macrocode}
\dim_set:Nn \col@sep
{ \bool_if:NTF \l_@@_tabular_bool \tabcolsep \arraycolsep }
\dim_compare:nNnTF \l_@@_tabular_width_dim = \c_zero_dim
{ \cs_set_nopar:Npn \@halignto { } }
{ \cs_set_nopar:Npx \@halignto { to \dim_use:N \l_@@_tabular_width_dim } }
\end{macrocode}

It \texttt{colortbl} is loaded, \texttt{\@tabarray} has been redefined to incorporate \texttt{CT@start}.
\begin{verbatim}
\end{verbatim}
\end{verbatim}

\begin{verbatim}
\l_@@_baseline_tl may have the value \texttt{t}, \texttt{c} or \texttt{b}. However, if the value is \texttt{b}, we compose the \texttt{array} (of \texttt{array}) with the option \texttt{t} and the right translation will be done further. Remark that \texttt{\str_if_eq:onTF} is fully expandable and we need something fully expandable here.
\begin{verbatim}
[ \str_if_eq:onTF \l_@@_baseline_tl c c t ]
\end{verbatim}

We keep in memory the standard version of \texttt{\ialign} because we will redefine \texttt{\ialign} in the environment \{NiceArrayWithDelims\} but restore the standard version for use in the cells of the array. However, since version 2.6a (version for the Tagging Project), \texttt{array} uses \texttt{\ar@ialign} instead of \texttt{\ialign}. In that case of \texttt{\ar@ialign}, of course, you do a saving of \texttt{\ar@ialign}.
\begin{verbatim}
\bool_if:NTF \c_@@_tagging_array_bool
{ \cs_set_eq:NN \@@_old_ar@ialign: \ar@ialign }
{ \cs_set_eq:NN \@@_old_ialign: \ialign }
\end{verbatim}

The following command creates a row node (and not a row of nodes!).
\begin{verbatim}
\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:
} }
\end{verbatim}

The \texttt{\hbox:n} (or \texttt{\hbox}) is mandatory.
\begin{verbatim}
\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 } }
\str_if_empty:NF \l_@@_name_str
{ \pgfnodealias
{ \l_@@_name_str-row\int_eval:n { \c@iRow + 1 } }
\endpgfpicture
} }
\pgfstoppicture
\endpgfpicture
} }
\end{verbatim}

42
The following must not be protected because it begins with \noalign.
\cs_new:Npn \@@_everycr: { \noalign { \@@_everycr_i: } }
\cs_new_protected:Npn \@@_everycr_i:
{ }
\bool_if:NT \c_@@_testphase_table_bool
\{ 
\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 those rules (the rules will be drawn by PGF).
\tl_if_empty:NF \l_@@_hlines_clist
\{ 
\tl_if_eq:NNF \l_@@_hlines_clist \c_@@_all_tl
\{ 
\exp_args:NNe
\clist_if_in:NnT \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:
\}
\cs_new_protected:Npn \@@_test_color_inside:
{ }
\bool_if:NT \l_@@_color_inside_bool
\{ 
\bool_if:NF \l_@@_color_inside_bool
\{ \@@_error:n { without~color-inside } \}
\}

We will issue an error only during the first run.
\bool_if:NF \g_@@_aux_found_bool
\{ \@@_error:n { without~color-inside } \}
}
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.

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` 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`.

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

---

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

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

\begin{verbatim}
\bool_if:NT \l_@@_small_bool {
  \cs_set_nopar:Npn \arraystretch { 0.47 }
  \dim_set:Nn \arraycolsep { 1.45 pt }
}
\end{verbatim}

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

\begin{verbatim}
\cs_set_eq:NN \@@_tuning_key_small: \scriptstyle
\end{verbatim}

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

\begin{verbatim}
\bool_if:NTF \c_@@_tagging_array_bool {
  \cs_set_nopar:Npn \ar@ialign
  {\bool_if:NT \c_@@_testphase_table_bool \tbl_init_cell_data_for_table:
    \@@_redefine_everycr:
    \dim_zero:N \tabskip
    \@@_some_initialization:
  }
  \cs_set_eq:NN \ar@ialign \@@_old_ar@ialign:
  \halign
}
\end{verbatim}

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

\begin{verbatim}
{\cs_set_eq:NN \ar@ialign \@@_old_ar@ialign:
  \halign
}
\end{verbatim}

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

\begin{verbatim}
{\cs_set_nopar:Npn \ialign
  {\@@_redefine_everycr:
   \dim_zero:N \tabskip
    \@@_some_initialization:
    \cs_set_eq:NN \ialign \@@_old_ialign:
    \halign
  }
}
\end{verbatim}
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).

\begin{verbatim}
1604  \cs_set_eq:NN \@@_old_Ldots \ldots
1605  \cs_set_eq:NN \@@_old_cdots \cdots
1606  \cs_set_eq:NN \@@_old_vdots \vdots
1607  \cs_set_eq:NN \@@_old_ddots \ddots
1608  \cs_set_eq:NN \@@_old_iddots \iddots
1609  \bool_if:NTF \l_@@_standard_cline_bool
1610  { \cs_set_eq:NN \cline \@@_standard_cline }
1611  { \cs_set_eq:NN \cline \@@_cline }
1612  \cs_set_eq:NN \Ldots \@@_Ldots
1613  \cs_set_eq:NN \Cdots \@@_Cdots
1614  \cs_set_eq:NN \Vdots \@@_Vdots
1615  \cs_set_eq:NN \Ddots \@@_Ddots
1616  \cs_set_eq:NN \Iddots \@@_Iddots
1617  \cs_set_eq:NN \Hline \@@_Hline:
1618  \cs_set_eq:NN \Hspace \@@_Hspace:
1619  \cs_set_eq:NN \Hdotsfor \@@_Hdotsfor:
1620  \cs_set_eq:NN \Vdotsfor \@@_Vdotsfor:
1621  \cs_set_eq:NN \Block \@@_Block:
1622  \cs_set_eq:NN \rotate \@@_rotate:
1623  \cs_set_eq:NN \OnlyMainNiceMatrix \@@_OnlyMainNiceMatrix:n
1624  \cs_set_eq:NN \multicolumn \@@_multicolumn:nnn
1625  \hook_gput_code:nnn { env / tabular / begin } { nicematrix }
1626  \cs_set_eq:NN \multicolumn \@@_old_multicolumn
1627  \@@_revert_colortbl:
\end{verbatim}

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:.

\begin{verbatim}
1640  \tl_if_exist:NT \l_@@_note_in_caption_tl
1641  { \tl_if_empty:NF \l_@@_note_in_caption_tl
1642  { \int_gset_eq:NN \g_@@_notes_caption_int \l_@@_note_in_caption_tl
1643  \int_gset:Nn \c@tabularnote { \l_@@_note_in_caption_tl } }
1644  { \int_gset_eq:NN \l_@@_first_row_int > \c_zero_int
1645  \int_gset_eq:NN \l_@@_last_row_int < \c_zero_int
1646  { \cs_set_eq:NN \multicolumn \@@_multirow-multicolumn
1647  \@@_revert_colortbl: }
1648  \bool_if:NT \l_@@_renew_dots_bool \@@_renew_dots:
\end{verbatim}

If there is one or several commands \tabularnote in the caption specified by the key \texttt{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!).

\begin{verbatim}
1644  \tl_if_exist:NT \l_@@_note_in_caption_tl
1645  { \tl_if_empty:NF \l_@@_note_in_caption_tl
1646  { \int_gset_eq:NN \g_@@_notes_caption_int \l_@@_note_in_caption_tl
1647  \int_gset:Nn \c@tabularnote { \l_@@_note_in_caption_tl } }
1650  \bool_if:NT \l_@@_renew_dots_bool \@@_renew_dots:
\end{verbatim}

The sequence \g_@@\_multicolumn\_cells\_seq will contain the list of the cells of the array where a command \texttt{\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).

\begin{verbatim}
\seq_gclear:N \g_@@_multicolumn_cells_seq
\seq_gclear:N \g_@@_multicolumn_sizes_seq
\end{verbatim}

The counter \texttt{\c@iRow} will be used to count the rows of the array (its incrementation will be in the first cell of the row).

\begin{verbatim}
\int_gset:Nn \c@iRow { \l_@@_first_row_int - 1 }
\end{verbatim}

At the end of the environment \{\texttt{\textbackslash array}\}, \texttt{\c@iRow} will be the total number de rows. \texttt{\g_@@\_row\_total\_int} will be the number or rows excepted the last row (if \texttt{\l_@@\_last\_row\_bool} has been raised with the option \texttt{last-row}).

\begin{verbatim}
\int_gzero_new:N \g_@@_row_total_int
\end{verbatim}

The counter \texttt{\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 \texttt{\g_@@\_col\_total\_int}. These counters are updated in the command \texttt{\@@\_cell\_begin:w} executed at the beginning of each cell.

\begin{verbatim}
\int_gzero_new:N \g_@@_col_total_int
\end{verbatim}

\begin{verbatim}
\cs_set_eq:NN \@ifnextchar \new@ifnextchar
\bool_gset_false:N \g_@@_last_col_found_bool
\end{verbatim}

During the construction of the array, the instructions \texttt{\textbackslash Cdots}, \texttt{\textbackslash Ldots}, etc. will be written in token lists \texttt{\g_@@\_Cdots\_lines_tl}, etc. which will be executed after the construction of the array.

\begin{verbatim}
\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
\end{verbatim}

This is the end of \texttt{\@@\_pre\_array\_ii:}.

The command \texttt{\@@\_pre\_array:} will be executed after analyse of the keys of the environment.

\begin{verbatim}
\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
\int_compare:nNnT \l_@@_last_row_int = { -1 }
{ \bool_set_true:N \l_@@_last_row_without_value_bool \int_gzero_new:N \g_@@_last_row_total_int
\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 } } }
\end{verbatim}

We recall that \texttt{\l_@@\_last\_row\_int} and \texttt{\l_@@\_last\_column\_int} are not the numbers of the last row and last column of the array. There are only the values of the keys \texttt{last-row} and \texttt{last-column} (maybe the user has provided erroneous values). The meaning of that counters does not change during the environment of \texttt{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).

\begin{verbatim}
\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 } } }
\end{verbatim}

47
If there is an exterior row, we patch a command used in \(@\_cell_begin:w\) in order to keep track of some dimensions needed to the construction of that “last row”.

\int_compare:NnT \l_@@_last_row_int > { -2 }

\tl_put_right:Nn \@@_update_for_first_and_last_row:

\dim_gset:Nn \l_@@_ht_last_row_dim
{ \mathdim:nn \l_@@_ht_last_row_dim { \box_ht:N \l_@@_cell_box } }
\dim_gset:Nn \l_@@_dp_last_row_dim
{ \mathdim:nn \l_@@_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 value of \g_@@_pos_of_blocks_seq has been written on the aux file and loaded before the (potential) execution of the \CodeBefore. Now, we clear that variable because it will be reconstructed during the creation of the array.

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 avoid 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 code in \@@_pre_array_ii: is used only here.
\@@_pre_array_ii:

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

box_clear_new:N \l_@@_the_array_box

We compute the width of both delimiters. We remind that, when the environment \{NiceArray\} is used, it’s possible to specify the delimiters in the preamble (eg \[ccc\]).

\dim_zero_new:N \l_@@_left_delim_dim
\dim_zero_new:N \l_@@_right_delim_dim

If the environment \{NiceArray\} is used, \verb|bBigg@| is a command of \texttt{amsmath}.

\box_set:Nn \l_tmpa_box { $ bBigg@ 5 \g_@@_left_delim_tl $ }
\dim_set:Nn \l_@@_left_delim_dim { \box_wd:N \l_tmpa_box }
\box_set:Nn \l_tmpa_box { $ bBigg@ 5 \g_@@_right_delim_tl $ }
\dim_set:Nn \l_@@_right_delim_dim { \box_wd:N \l_tmpa_box }

The command \verb|bBigg@| is a command of \texttt{amsmath}.
Here is the beginning of the box which will contain the array. The \hbox_set_end: corresponding to this \hbox_set:Nw will be in the second part of the environment (and the closing \c_math_toggle_token also).

\begin{verbatim}
\hbox_set:Nw \l_@@_the_array_box
\bool_if:NT \c_@@_testphase_table_bool
  { \UseTaggingSocket { tbl / hmode / begin } }
\skip_horizontal:N \l_@@_left_margin_dim
\skip_horizontal:N \l_@@_extra_left_margin_dim
\c_math_toggle_token
\bool_if:NTF \l_@@_light_syntax_bool
  { \use:c { @@-light-syntax } }
  { \use:c { @@-normal-syntax } }
\end{verbatim}

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

\begin{verbatim}
\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:NV \g_@@_pre_code_before_tl \l_tmpa_tl
  \bool_set_true:N \l_@@_code_before_bool
\end{verbatim}

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...

\begin{verbatim}
\@@_pre_array:
\end{verbatim}

10 The \CodeBefore

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

\begin{verbatim}
\cs_new_protected:Npn \@@_pre_code_before:
  \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_set_eq:NN \g_@@_row_total_int { \seq_item:Nn \g_@@_size_seq 3 }
  \int_set_eq:NN \g_@@_col_total_int { \seq_item:Nn \g_@@_size_seq 6 }
\end{verbatim}

Now, we will create all the col nodes and row nodes with the informations written in the aux file. You use the technique described in the page 1229 of pgfmanual.pdf, version 3.1.4b.

\begin{verbatim}
\pgfsys@markposition { \@@_env: - position }
\pgfsys@getposition { \@@_env: - position } \@@_picture_position:
\pgfpicture
\pgf@relevantforpicturesizefalse
\end{verbatim}
First, the recreation of the row nodes.

\begin{Verbatim}
\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: \}
\end{Verbatim}

Now, the recreation of the col nodes.

\begin{Verbatim}
\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: \}
\end{Verbatim}

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

\begin{Verbatim}
\@@_create_diag_nodes:
\end{Verbatim}

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

\begin{Verbatim}
\bool_if:NT \g_@@_recreate_cell_nodes_bool \@@_recreate_cell_nodes:
\end{Verbatim}

Now, the recreation of the nodes of the blocks \textit{which have a name}.

\begin{Verbatim}
\@@_create_blocks_nodes:
\IfPackageLoadedTF \{ tikz \}
\{
\tikzset
\{
\begin{Verbatim}
\every-picture / .style = \{ overlay , name-prefix = \@@_env: - \}
\end{Verbatim}
\}
\}\end{Verbatim}

\begin{Verbatim}
\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
\end{Verbatim}

\begin{Verbatim}
\cs_new_protected:Npn \@@_exec_code_before:
\{
\seq_gclear_new:N \g_@@_colors_seq
\The sequence \g_@@_colors_seq will always contain as first element the special color \texttt{nocolor}: when that color is used, no color will be applied in the corresponding cells by the other coloring commands of \texttt{nicematrix}.

\begin{Verbatim}
\@@_add_to_colors_seq:nn \{ \texttt{nocolor} \} \{
\bool_gset_false:N \g_@@_recreate_cell_nodes_bool
\group_begin:
\end{Verbatim}

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

\begin{Verbatim}
\bool_if:NT \l_@@_tabular_bool \c_math_toggle_token
\end{Verbatim}

50
The following code is a security for the case the user has used babel with the option spanish: in that case, the characters < (de code ascii 60) and > are activated and Tikz is not able to solve the problem (even with the Tikz library babel).

\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:NV \@@_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
\bool_if:NT \l_@@_tabular_bool \c_math_toggle_token
\group_end:
\bool_if:NT \g_@@_recreate_cell_nodes_bool
\{ \tl_put_left:Nn \@@_node_for_cell: \@@_patch_node_for_cell: \}

\keys_define:nn \{ NiceMatrix / CodeBefore \}
\{ create-cell-nodes .bool_gset:N = \g_@@_recreate_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.
\ca_new_protected:Npn \@@_CodeBefore:w \#1 \q_stop
\{ \bool_if:NT \g_@@_aux_found_bool
\{ \@@_pre_code_before:
\#1
\}
\}

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.
\ca_new_protected:Npn \@@_recreate_cell_nodes:
\{ \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int

51
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.\footnote{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).}
11 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 \vgroup (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.

\vgroup

\tl_gset:Nn \g_@@_left_delim_tl { \sl } \tl_gset:Nn \g_@@_right_delim_tl { \sr } \tl_gset:Nn \g_@@_user_preamble_tl { \sp } \tl_if_empty:NT \g_@@_user_preamble_tl { \@@_fatal:n { empty~preamble } } \tl_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

\int_gzero:N \g_@@_block_box_int \dim_zero:N \g_@@_width_last_col_dim \dim_zero:N \g_@@_width_first_col_dim \bool_gset_false:N \g_@@_row_of_col_done_bool \str_if_empty:NT \g_@@_name_env_str \@@_fatal:n { 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 53
The command \CT@arc@ contains the instruction of color for the rules of the array\textsuperscript{7}. This command is used by \CT@arc@ but we use it also for compatibility with colortbl. But we want also to be able to use color for the rules of the array when colortbl is not loaded. That’s why we do the following instruction which is in the patch of the beginning of arrays done by colortbl. Of course, we restore the value of \CT@arc@ at the end of our environment.

\cs_gset_eq:NN \@@_old_CT@arc@ \CT@arc@

We deactivate Tikz externalization because we will use PGF pictures with the options overlay and remember picture (or equivalent forms). We deactivate with \tikzexternaldisable and not with \tikzset{external/export=false} which is not equivalent.

\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 (except the blocks with the key hvlines).
\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 informations written in the aux file during previous compilations corresponding to the current environment.
\tl_if_exist:cTF { c_@@ _ \int_use:N \g_@@_env_int _ tl } \{ \bool_gset_true:N \g_@@_aux_found_bool \use:c { c_@@ _ \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:HF \g_@@_code_before_tl
\{ \bool_set_true:N \l_@@_code_before_bool \tl_put_right:NV \l_@@_code_before_tl \g_@@_code_before_tl \}
\tl_if_empty:HF \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 \}

\textsuperscript{7}e.g. \color[rgb]{0.5,0.5,0}
The argument \#6 is the last argument of \texttt{\{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 \texttt{\@@\_CodeBefore\_Body:w}. After that job, the command \texttt{\@@\_CodeBefore\_Body:w} will go on with \texttt{\@@\_pre\_array:}.

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

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

\end{verbatim}

End of the construction of the array (in the box \texttt{\l_@@\_the\_array\_box}).

If the user has used the key width without any column X, we raise an error.

\begin{verbatim}
\bool_if:NT \l_@@_width_used_bool
  \int_if_zero:nT \g_@@_total_X_weight_int
    \@@_error_or_warning:n { width\-without\-X\-columns }

\end{verbatim}

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, \texttt{\l_@@\_X\_columns\_dim} will be the width of a column of weight 1. For a X-column of weight \(n\), the width will be \texttt{\l_@@\_X\_columns\_dim} multiplied by \(n\).

\begin{verbatim}
\int_compare:nNnT \g_@@_total_X_weight_int > \c_zero_int

\end{verbatim}
( \l_@@_width_dim - \box_wd:N \l_@@_the_array_box )
/ \int_use:N \g_@@_total_X_weight_int
+ \l_@@_X_columns_dim
}
}
}
}

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 > \c@iRow
{
\bool_if:NF \l_@@_last_row_without_value_bool
{ \int_compare:nNnF \l_@@_last_row_int = \c@iRow
{ \@@_error:n { Wrong last-row }
\int_gset_eq:NN \l_@@_last_row_int \c@iRow
}
}
}

Now, the definition of \c@jCol and \g_@@_col_total_int change: \c@jCol will be the number of columns without the “last column”; \g_@@_col_total_int will be the number of columns with this “last column”.

\int_gset_eq:NN \c@jCol \g_@@_col_total_int
\bool_if:nTF { ! \g_@@_delims_bool }
{ \tl_if_eq:NNTF \l_@@_baseline_tl \c_@@_c_tl
\@@_use_arraybox_with_notes_c:
{ \tl_if_eq:NNTF \l_@@_baseline_tl \c_@@_b_tl
\@@_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:nT \l_@@_first_row_int
{ \dim_set_eq:NN \l_tmpa_dim \g_@@_dp_row_zero_dim
}

---

8We remind that the potential “first column” (exterior) has the number 0.
We compute $\l_{\text{tmpb_dim}}$ which is the total height of the “last row” below the array (when the key `last-row` is used). A value of $-2$ for $\l_{\text{@@_last_row_int}}$ means that there is no “last row”.\footnote{A value of $-1$ for $\l_{\text{@@_last_row_int}}$ means that there is a “last row” but the user have not set the value with the option `last row` (and we are in the first compilation).} $\\int_{\text{compare:NnTF}} \l_{\text{@@_last_row_int}} > \{-2\}$ $\\{ \begin{array}{l} \\dim_{\text{set_eq:NW}} \l_{\text{tmpb_dim}} \g_{\text{@@_ht_last_row_dim}} \\
\\dim_{\text{add:Nn}} \l_{\text{tmpb_dim}} \g_{\text{@@_dp_last_row_dim}} \\
\end{array} \} \\{ \dim_{\text{zero:N}} \l_{\text{tmpb_dim}} \}$ $\hbox_{\text{set:Nn}} \l_{\text{tmpa_box}} \{ \c_{\text{math-toggle_token}} \l_{\text{@@_color:o delimiters_color_tl}} \exp_{\text{after:WN}} \left \g_{\text{@@_left_delim_tl}} \vcenter \{ \begin{array}{l}
\text{We take into account the “first row” (we have previously computed its total height in $\l_{\text{tmpa_dim}}$). The $\hbox:n$ (or $\hbox$) is necessary here.}
\skip_{\text{vertical:n}} \{-\l_{\text{tmpa_dim}} - \text{arrayrulewidth}\} \\
\hbox \\
\{ \\bool_{\text{if:NTF}} \l_{\text{@@_tabular_bool}} \\
\{ \\skip_{\text{horizontal:N}} -\text{tabcolsep} \} \\
\{ \\skip_{\text{horizontal:N}} -\text{arraycolsep} \} \\
\@@_{\text{use_arraybox_with_notes_c}}: \\
\bool_{\text{if:NTF}} \l_{\text{@@_tabularBool}} \\
\{ \\skip_{\text{horizontal:N}} -\text{tabcolsep} \} \\
\{ \\skip_{\text{horizontal:N}} -\text{arraycolsep} \} \\
\} \end{array} \} \vcenter \} \\} \\
\exp_{\text{after:WN}} \right \g_{\text{@@_right_delim_tl}} \c_{\text{math-toggle_token}} \}$ $\text{We take into account the “last row” (we have previously computed its total height in $\l_{\text{tmpb_dim}}$).}$ $\skip_{\text{vertical:N}} \{-\l_{\text{tmpb_dim}} \}$ $\skip_{\text{vertical:N}} \text{arrayrulewidth}$ $\} \\exp_{\text{after:WN}} \right \g_{\text{@@_right_delim_tl}} \c_{\text{math-toggle_token}} \}$ $\text{Now, the box $\l_{\text{tmpa_box}}$ is created with the correct delimiters.}$ $\text{We will put the box in the TeX flow. However, we have a small work to do when the option}$ \text{delimiters/max-width} is used. $\\bool_{\text{if:NTF}} \l_{\text{@@_delimiters_max_width_bool}} \{ \\skip_{\text{box_in_flow:bis:nn}} \g_{\text{@@_left_delim_tl}} \g_{\text{@@_right_delim_tl}} \} \\skip_{\text{box_in_flow:}} \}$ $\text{We take into account a potential “last column” (this “last column” has been constructed in an}$ overlapping position and we have computed its width in $\g_{\text{@@_width_last_col_dim}}$: see p. 90). $\\bool_{\text{if:NT}} \g_{\text{@@_last_col_found_bool}} \{ \\skip_{\text{horizontal:N}} \g_{\text{@@_width_last_col_dim}} \} \\bool_{\text{if:NT}} \l_{\text{@@_preamble_bool}} \{ \\int_{\text{compare:NnT}} \\{ \\text{columns-not-used} \} \}$
The aim of the following \texttt{egroup} (the corresponding \texttt{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.

\texttt{egroup}

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

\texttt{\iow_now:Nn \@mainaux { \ExplSyntaxOn }}
\texttt{\iow_now:Nn \@mainaux { \char_set_catcode_space:n { 32 } }}
\texttt{\iow_now:Nx \@mainaux}
\texttt{\tl_gset:cn { c_@@_ \int_use:N \g_@@_env_int _ tl }}
\texttt{\exp_not:o \g_@@_aux_tl}
\texttt{\iow_now:Nn \@mainaux { \ExplSyntaxOff }}
\texttt{\bool_if:NT \g_@@_footnote_bool \endsavenotes}

This is the end of the environment \texttt{NiceArrayWithDelims}.

\section{We construct the preamble of the array}

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

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

\texttt{\cs_new_protected:Npn \@@_transform_preamble:}
\texttt{\{ \@@_transform_preamble_i: \@@_transform_preamble_ii: \}}
\texttt{\cs_new_protected:Npn \@@_transform_preamble_i:}
\texttt{\{ \int_gzero:N \c@jCol \}}
\texttt{\seq_gclear:N \g_@@_cols_vlism_seq}
\texttt{\g_tmpb_bool \bool_gset_false:N \g_tmpb_bool}
\texttt{\tl_gclear_new:N \g_@@_array_preamble_tl}
\texttt{\tl_if_eq:NNTF \l_@@_vlines_clist \c_@@_all_tl}
\texttt{\tl_gset:Nn \g_@@_array_preamble_tl \skip_horizontal:N \arrayrulewidth}
\texttt{\bool_gset_false:N \g_tmpb_bool}
\texttt{\seq_gclear:N \g_@@_cols_vlism_seq}
\texttt{\bool_gset_false:N \g_tmpb_bool}
\texttt{\tl_gclear_new:N \g_@@_pre_cell_tl}
\texttt{\tl_gclear:N \l_@@_vlines_clist \c_@@_all_tl}
\texttt{\tl_if_eq:NNTF \l_@@_vlines_clist 1}
\texttt{\tl_gset:Nn \g_@@_array_preamble_tl \skip_horizontal:N \arrayrulewidth}
\texttt{\tl_gclear:N \l_@@_vlines_clist 1}
\texttt{\tl_gset:Nn \g_@@_array_preamble_tl \}}
\texttt{\}}

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

The following sequence will store the arguments of the successive \texttt{>} in the preamble.

\texttt{\tl_gclear:N \g_@@_pre_cell_tl}
\texttt{\tl_gclear:N \l_@@_vlines_clist \c_@@_all_tl}
\texttt{\tl_if_eq:NNTF \l_@@_vlines_clist 1}
\texttt{\tl_gset:Nn \g_@@_array_preamble_tl \skip_horizontal:N \arrayrulewidth}
\texttt{\tl_gclear:N \l_@@_vlines_clist 1}
\texttt{\tl_gset:Nn \g_@@_array_preamble_tl}
\arrayrulewidth \{ \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 : NV \@rec_preamble : n \g@user_preamble_tl \stop
\int_gset_eq : NN \g@static_num_of_col_int \c@jCol

\@replace_columncolor:

\hook_gput_code : nnn \{ begindocument \} { . }
\{ \IfPackageLoadedTF \{ colortbl \} \{ \}

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

\regex_const : Nn \c@columncolor_regex \{ \c \{ \ttt{\columncolor} \} \}
\cs_new_protected : Npn \@replace_columncolor : \
\{ \regex_replace_all : NN \c@columncolor_regex \{ \c \{ \@columncolor_preamble \} \}
\g@array_preamble_tl
\}

\cs_new_protected : Npn \@replace_columncolor : 
\{ \cs_set_eq : NN \columncolor \@columncolor_preamble \}
\}

\cs_new_protected : Npn \@transform_preamble_ii : 
\{

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

\bool_if : NT \g@tmpb_bool \{ \bool_set_true : N \l@@bar_at_end_of_pream_bool \}

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

\bool_if : NT \g@delims_bool \{ \bool_set_true : N \l@@delims_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@array_preamble_first_col_tl \}
\{ \bool_if : NF \g@delims_bool \\
\{ \bool_if : NF \l@tabular_bool \\
\{ \tl_if_empty : NT \l@vlines_clist 
\}
We add a last column to raise a good error message when the user puts more columns than allowed by its preamble. However, for technical reasons, it’s not possible to do that in \{NiceTabular*\} (we control that with the value of \l_@@_tabular_width_dim).}

\dim_compare:nNnTF \l_@@_tabular_width_dim = \c_zero_dim
\{ \tl_gput_right:Nn \g_@@_array_preamble_tl
\{ > { \@@_error_too_much_cols: } l \}
\}

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.\
\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:NV \@@_rec_preamble:n \l_tmpb_tl
\}
\}
\}

\@_{10}\text{We do that because it’s an easy way to insert the letter at some places in the code that we will add to }\g_@@_array_preamble_tl.
For \( c, l \) and \( r \)

\begin{verbatim}
\cs_new:Npn \@@_c #1
{\tl_gput_right:N\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:w c < \@@_cell_end: }
 \int_gincr:N\c@jCol
 \@@_rec_preamble_after_col:n}
\cs_new:Npn \@@_l #1
{\tl_gput_right:N\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:w \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:Npn \@@_r #1
{\tl_gput_right:N\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:w \tl_set_eq:NN \l_@@_hpos_cell_tl \c_@@_r_tl }
 r < \@@_cell_end: }
 \int_gincr:N\c@jCol
 \@@_rec_preamble_after_col:n}
\cs_new:Npn \@@_! #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 !}
\cs_new:cpn{@@_|}{@@_#1}{\use:c{@@_|} |}{\@@_make_preamble_i_ii:nn { } #1}
\cs_new_protected:Npn \@@_make_preamble_i_ii:n #1
{\str_if_eq:nnTF{#1}{|}{\@@_make_preamble_i_ii:nn { } #1}{\@@_make_preamble_i_ii:nn { } #1}}
\cs_new:cpn{@@_|}{@@_#1}{|}{\@@_make_preamble_i_ii:nn { } #1}
\cs_new:cpn{@@|}{@@_#1}{|}{\@@_make_preamble_i_ii:nn { } #1}
\cs_new:cpn{@@\_}{@@_#1}{|}{\@@_make_preamble_i_ii:nn { } #1}
\cs_new:cpn{@@\_!}{@@_#1}{!}{\@@_make_preamble_i_ii:nn { } #1}
\cs_new:cpn{@@\_|}{@@_#1}{|}{\@@_make_preamble_i_ii:nn { } #1}
\cs_new:cpn{@@\_@}{@@_#1}{@}{\@@_make_preamble_i_ii:nn { } #1}
\end{verbatim}

For \(!\) and \(@\)

\begin{verbatim}
\cs_new:cpn{@@_ \token_to_str:N ! }{#1 \#2}
{\tl_gput_right:Nn\g_@@_array_preamble_tl \g_@@_pre_cell_tl
 \tl_gclear:N\g_@@_pre_cell_tl
 \tl_gput_right:Nn\g_@@_array_preamble_tl
 \{ #1 \{ #2 \} }
 \@@_rec_preamble:n}
\cs_new:cpn{@@_ \token_to_str:N @ }{@@_ \token_to_str:N ! }
\cs_set_eq:cc{@@_|}{@@_#1}{\use:c{@@_|} |}{\@@_make_preamble_i_i:n #1}
\end{verbatim}

For \( | \)

\begin{verbatim}
\cs_new: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
{\str_if_eq:nnTF{#1}{|}{\@@_make_preamble_i_i:n #1}{\@@_make_preamble_i_ii:nn { } #1}}
\end{verbatim}
Here, the command \texttt{\textcolor{red}{dim\_eval:n}} is mandatory. We don’t have provided value for 	exttt{start} nor for 	exttt{end}, which means that the rule will cover (potentially) all the rows of the array.

The specifier \texttt{p} (and also the specifiers \texttt{m}, \texttt{b}, \texttt{V} and \texttt{X}) have an optional argument between square brackets for a list of \textit{key-value} pairs. Here are the corresponding keys.

```latex
\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 ,
  R .code:n = \IfPackageLoadedTF { ragged2e } { \str_set_eq:NN \l_@@_hpos_col_str \c_@@_R_str } ,
  R .value_forbidden:n = true ,
  L .code:n =
}
```

The specifier \texttt{p} (and also the specifiers \texttt{m}, \texttt{b}, \texttt{V} and \texttt{X}) have an optional argument between square brackets for a list of \textit{key-value} pairs. Here are the corresponding keys.
\IfPackageLoadedTF { ragged2e }
\{ \str_set_eq:NN \l_@@_hpos_col_str \c_@@_L_str \}
\{ \@@_error_or_warning:n { ragged2e-not-loaded } \str_set_eq:NN \l_@@_hpos_col_str \c_@@_l_str \}
L.value_forbidden:n = true ,
C.code:n = \IfPackageLoadedTF { ragged2e }
\{ \str_set_eq:NN \l_@@_hpos_col_str \c_@@_C_str \}
\{ \@@_error_or_warning:n { ragged2e-not-loaded } \str_set_eq:NN \l_@@_hpos_col_str \c_@@_c_str \}
C.value_forbidden:n = true ,
S.code:n = \str_set_eq:NN \l_@@_hpos_col_str \c_@@_si_str ,
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 \texttt{p} but also \texttt{b} and \texttt{m}.
\cs_new:Npn \@@_p #1
\{ \str_set:Nn \l_@@_vpos_col_str { #1 } \}
\cs_set_eq:NN \@@_b \@@_p
\cs_set_eq:NN \@@_m \@@_p
\cs_new_protected:Npn \@@_make_preamble_ii_i:n #1
\{ \str_if_eq:nnTF { #1 } { [ } { \@@_make_preamble_ii_ii:w [ \] { #1 } } \}
\cs_new_protected:Npn \@@_make_preamble_ii_ii:w \[ #1 ]
\{ \@@_make_preamble_ii_iii:nn { #1 } \}
\cs_new_protected:Npn \@@_make_preamble_ii_iii:nn #1 #2 #3
\{ \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.
The parameter \_\_hpos_col_str (as \_\_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 \_\_hpos_cell_tl which will provide the horizontal alignment of the column to which belongs the cell.

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

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

#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 \_\_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.
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 }
\bool_if:NT \c_@@_testphase_table_bool 
{ \tag_struct_begin:n { tag = Div } }
\@@_cell_begin:w
We use the form \minipage–\endminipage (\varwidth–\endvarwidth) for compatibility with colcell
(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 { } 
\} 
\bool_if:NT \c_@@_testphase_table_bool \tagpdfparaOn
Now, the potential code for the horizontal position of the content of the cell (\centering, \raggedright, \RaggedRight, etc.).
\#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: 
\bool_if:NT \c_@@_testphase_table_bool \tag_struct_end:
}\}
\}
\str_new:N \c_@@_ignorespaces_str 
\str_set:Nx \c_@@_ignorespaces_str { \ignorespaces }
\str_remove_all:Nn \c_@@_ignorespaces_str { - }

In order to test whether a cell is empty, we test whether it begins by \ignorespaces\unskip. However,
in some circumstances, for example when \collectcell of colcell is used, the cell does not
begin with \ignorespaces. In that case, we consider as not empty...
First, we test if the next token is \ignorespaces and it’s not very easy...
\cs_new_protected:Npn \@@_test_if_empty: { \peek_after:Nw \@@_test_if_empty_i: }
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.

By putting instructions in $\g_\text{cell_after_hook_tl}$, we require a post-action of the box $\l_\text{cell_box}$.

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).

For V (similar to the V of varwidth).
For \texttt{w} and \texttt{W}

\begin{verbatim}
\cs_new:Npn \@@_w { \@@_make_preamble_w:nnnn { } }
\cs_new:Npn \@@_W { \@@_make_preamble_w:nnnn { \@@_special_W: } }
#1 is a special argument: empty for \texttt{w} and equal to \texttt{\@special_W:} for \texttt{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.
\end{verbatim}

First, the case of an horizontal alignment equal to \texttt{s} (for \textit{stretch}).
#1 is a special argument: empty for \texttt{w} and equal to \texttt{\@special_W:} for \texttt{W};
#2 is the width of the column.

\begin{verbatim}
\cs_new_protected:Npn \@@_make_preamble_w:i:nnn #1 #2 #3 #4
\str_if_eq:nnTF { #3 } { s } { \@@_make_preamble_w:i:nnn { #1 } { #2 } { #3 } { #4 } }
\end{verbatim}

\begin{verbatim}
\str_if_eq:nnTF { #2 } { [ } \@@_make_preamble_V:i:w [ ] { #2 }
\end{verbatim}
Then, the most important version, for the horizontal alignments types of \texttt{c}, \texttt{l} and \texttt{r} (and not \texttt{s}).

The parameter \texttt{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 increment the counter of columns and then we test for the presence of a \texttt{<}.

For \texttt{S} (of \texttt{siunitx}).
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
\}
\@@_fatal:n { siunitx-not-loaded }
\}

For (, [ and \{.
\cs_new: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
\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 }
\}
\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:Nx \g_@@_pre_code_after_tl { \@@_delimiter:nnn #1 \int_eval:n { \c@jCol + 1 } \c_true_bool }
\tl_if_in:nnTF { ( \[ \{ ) \] \} } { #2 }
\{ \@@_error:nn { delimiter~after~opening } { #2 }
\@@@@rec_preamble:n
\}
\{ \@@_rec_preamble:n #2 
\}
\cs_set_eq:cc { \token_to_str:N \{ } { \token_to_str:N ( }
\cs_new:cpn { \token_to_str:N \{ } { \token_to_str:N ( }

In fact, if would be possible to define \left and \right as no-op.
\cs_new:cpn { \token_to_str:N \left } #1 { \use:c { \token_to_str:N \left } }
\cs_new:cpn { \token_to_str:N \right } #1 { \use:c { \token_to_str:N \right } }

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:cpn { \token_to_str:N \right } #1 { \use:c { \token_to_str:N \right } }
\bool_if:NT \l_@@_small_bool { \@@_fatal:n { Delimiter-with-small } }
\tl_if_in:nTF \{ [ \} \} \left \right { #2 }
\{ \@@_make_preamble_v:nnn #1 #2 }
\{ \@@_rec_preamble:n #2 
\}
\{ \@@_rec_preamble:n #2 
\}
\cs_set_eq:cc { \token_to_str:N \{ } { \token_to_str:N ( }
\cs_new:cpn { \token_to_str:N \{ } { \token_to_str:N ( }

69
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 @}...).

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

70
\begin{verbatim}
\exp_args:NNe \clist_if_in:NnTF \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
\}
\cs_new_protected:Npn \@@_rec_preamble_after_col_ii:n #1
\{
  \tl_if_eq:NNTF \l_@@_vlines_clist \c_@@_all_tl
  { \tl_gput_right:Nn \g_@@_array_preamble_tl
    { @ { #1 \skip_horizontal:N \arrayrulewidth } }
  }{ 
    \exp_args:NNe \clist_if_in:NnTF \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: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
\}
\cs_new: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_i:n { #1 }
\cs_new:cpn { @@_\token_to_str:N \NC@find } #1 { \@@_rec_preamble:n }
\cs_new: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
\}
\end{verbatim}

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.

\begin{verbatim}
\cs_new_protected:Npn \@@_rec_preamble_after_col_i:n #1
\{
  \tl_if_eq:NNTF \l_@@_vlines_clist \c_@@_all_tl
  { \tl_gput_right:Nn \g_@@_array_preamble_tl
    { < { #1 } }
  }{ 
    \exp_args:NNe \clist_if_in:NnTF \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: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
\}
\end{verbatim}

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

\begin{verbatim}
\cs_new:cpn { @@_ } #1 #2 \NC@find \cs_new:cpn { @@_ } #1 \NC@find \cs_new:cpn { @@_ } #1 \NC@find
\end{verbatim}

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.

\begin{verbatim}
\cs_new:Npn \@@_X #1 #2
\{
  \str_if_eq:nnTF { #2 } { \#2 } { \} } \cs_new:cpn { @@_X } #1 \cs_new:cpn { @@_X } #1 \cs_new:cpn { @@_X } #1
\end{verbatim}
is the optional argument of the \(X\) specifier (a list of key-value pairs).

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

\[
\begin{align*}
\keys_define:nn \{ \text{nicematrix / X-column} \} \\
\{ \text{unknown .code:n = \int_set:Nn \_\_\_weight_int \{ \_\_\keys_key_str \} } \\
\end{align*}
\]

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 \(\_\_\_hpos_col_str\) 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 \(X\)).

\[
\str_set:Nn \_\_\_hpos_col_str \{ j \}
\]

The possible values of \(\_\_\_vpos_col_str\) are \(p\) (the initial value), \(m\) and \(b\) (when the user has used the corresponding key in the optional argument of the specifier \(X\)).

\[
\str_set:Nn \_\_\_vpos_col_str \{ p \}
\]

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

\[
\int_zero_new:N \_\_\_weight_int \\
\int_set_eq:NN \_\_\_weight_int \c_one_int \\
\_\_\_keys_p_column:n \{ #1 \}
\]

The unknown keys are put in \_\_\_tmpa_tl

\[
\keys_set:no \{ \text{nicematrix / X-column} \} \_\_\_tmpa_tl \\
\int_compare:nNnT \_\_\_weight_int < \_\_\_\_zero_int \\
\_\_\_error_or_warning:n \{ \text{negative-weight} \} \\
\int_set:Nn \_\_\_weight_int \{ - \_\_\_weight_int \}
\]

\[
\int_gadd:Nn \_\_\_total_X_weight_int \_\_\_weight_int
\]

We test whether we know the width of the \(X\)-columns by reading the aux file (after the first compilation, the width of the \(X\)-columns is computed and written in the aux file).

\[
\bool_if:NTF \_\_\_X_columns_aux_bool \\
\exp_args:Nne \_\_\_\_make_preamble_ii_iv:nnn \\
\{ \_\_\_weight_int \_\_\_X_columns_dim \} \\
\{ \text{minipage} \} \\
\{ \_\_\_\_no_update_width: \} \\
\}
\]

\[
\_\_\_\_\_gput_right:Nn \_\_\_\_array_preamble_tl \\
\{ \}
\]

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 \texttt{\NotEmpty}.

\[
\NotEmpty
\]

The following code will nullify the box of the cell.

\[
\_\_\_\_\_\_\_\_\_gput_right:Nn \_\_\_\_\_cell_after_hook_tl \\
\{ \text{hbox_set:Nn \_\_\_\_\_cell_box } \}
\]
We put a \{minipage\} to give to the user the ability to put a command such as \texttt{\centering} in the \RowStyle.

\begin{minipage}{5 cm}
\texttt{arraybackslash}
\end{minipage}
\begin{minipage}{5 cm}
c
\end{minipage}
\begin{minipage}{5 cm}
\texttt{\@\_cell\_end:}
\end{minipage}
\begin{minipage}{5 cm}
\texttt{\int\_gincr:N \c@jCol}
\end{minipage}
\begin{minipage}{5 cm}
\texttt{\@\_rec\_preamble\_after\_col:n}
\end{minipage}
\begin{minipage}{5 cm}
\texttt{\cs\_new\_protected:Npn \@@\_no\_update\_width:}
\end{minipage}
\begin{minipage}{5 cm}
\texttt{\tl\_gput\_right:Nn \g\_@@\_cell\_after\_hook\_tl}
\end{minipage}
\begin{minipage}{5 cm}
\texttt{\cs\_set\_eq:NN \@@\_update\_max\_cell\_width: \@\_do\_nothing:}
\end{minipage}
\begin{minipage}{5 cm}
\texttt{\@@\_rec\_preamble:n}
\end{minipage}

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

\begin{minipage}{5 cm}
\texttt{\cs\_new\_protected:Npn \@@\_make\_preamble\_vlism:n \#1}
\end{minipage}
\begin{minipage}{5 cm}
\texttt{\seq\_gput\_right:Nx \g\_@@\_cols\_vlism\_seq}
\end{minipage}
\begin{minipage}{5 cm}
\texttt{\int\_eval:n \{ \c@jCol + 1 \}}
\end{minipage}
\begin{minipage}{5 cm}
\texttt{\tl\_gput\_right:Nx \g\_@@\_array\_preamble\_tl}
\end{minipage}
\begin{minipage}{5 cm}
\texttt{\exp\_not:N \{ \skip\_horizontal:N \array\_rule\_width \}}
\end{minipage}
\begin{minipage}{5 cm}
\texttt{\@@\_rec\_preamble:n}
\end{minipage}

The token $\texttt{\textbackslash stop}$ 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.

\begin{minipage}{5 cm}
\texttt{\cs\_set\_eq:CN \{ \texttt{\textbackslash token\_to\_str:N} \texttt{\textbackslash stop} \} \texttt{\use\_none:n}}
\end{minipage}

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

\begin{minipage}{5 cm}
\texttt{\cs\_new\_protected:cpn \{ \texttt{\textbackslash token\_to\_str:N} \texttt{\textbackslash hline} \}}
\end{minipage}
\begin{minipage}{5 cm}
\texttt{\texttt{\textbackslash token\_to\_str:N} \texttt{\hline}}
\end{minipage}
\begin{minipage}{5 cm}
\texttt{\texttt{\textbackslash token\_to\_str:N} \texttt{\textbackslash hline}}
\end{minipage}
\begin{minipage}{5 cm}
\texttt{\texttt{\textbackslash token\_to\_str:N} \texttt{\textbackslash hline}}
\end{minipage}
\begin{minipage}{5 cm}
\texttt{\texttt{\textbackslash token\_to\_str:N} \texttt{\textbackslash hline}}
\end{minipage}
\begin{minipage}{5 cm}
\texttt{\texttt{\textbackslash token\_to\_str:N} \texttt{\textbackslash hline}}
\end{minipage}

\section{The redefinition of $\texttt{\textbackslash multicolumn}$}

The following command must \textit{not} be protected since it begins with $\texttt{\textbackslash multispan}$ (a TeX primitive).

\begin{minipage}{5 cm}
\texttt{\cs\_new:Npn \@@\_multicolumn:nnn \#1 \#2 \#3}
\end{minipage}
\begin{minipage}{5 cm}
\texttt{\multispan \{ \#1 \}}
\end{minipage}
\begin{minipage}{5 cm}
\texttt{\texttt{\textbackslash token\_to\_str:N} \texttt{\textbackslash hline}}
\end{minipage}
\begin{minipage}{5 cm}
\texttt{\texttt{\textbackslash token\_to\_str:N} \texttt{\textbackslash hline}}
\end{minipage}
\begin{minipage}{5 cm}
\texttt{\texttt{\textbackslash token\_to\_str:N} \texttt{\textbackslash hline}}
\end{minipage}
\begin{minipage}{5 cm}
\texttt{\texttt{\textbackslash token\_to\_str:N} \texttt{\textbackslash hline}}
\end{minipage}
\begin{minipage}{5 cm}
\texttt{\texttt{\textbackslash token\_to\_str:N} \texttt{\textbackslash hline}}
\end{minipage}

The following lines are from the definition of $\texttt{\textbackslash multicolumn}$ in \texttt{array} (and \textit{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 $\texttt{\textbackslash multicolumn}$.
Now, we patch the (small) preamble as we have done with the main preamble of the array.

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

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

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

We add some lines.

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.
For \texttt{c}, \texttt{l} and \texttt{r}
\begin{verbatim}
\cs_new_protected:Npn \@@_make_m_preamble_i:n #1
{ \tl_gput_right:Nn \g_@@_preamble_tl
{ \@@_cell_begin:w \cs_set_nopar:Npn \l_@@_hpos_cell_tl { #1 } }
#1 \@@_cell_end:
}
\end{verbatim}
We test for the presence of a <.
\begin{verbatim}
\@@_make_m_preamble_x:n
\end{verbatim}
For >, ! and @
\begin{verbatim}
\cs_new_protected:Npn \@@_make_m_preamble_ii:nn #1 #2
{ \tl_gput_right:Nn \g_@@_preamble_tl
{ \@@_cell_begin:w \cs_set_nopar:Npn \l_@@_hpos_cell_tl { #1 } }
#1 { #2 } \@@_cell_end:
}
\end{verbatim}
For |
\begin{verbatim}
\cs_new_protected:Npn \@@_make_m_preamble_iii:n #1
{ \tl_gput_right:Nn \g_@@_preamble_tl
{ \@@_cell_begin:w \cs_set_nopar:Npn \l_@@_hpos_cell_tl { #1 } }
\@@_cell_end:
}
\end{verbatim}
For p, m and b
\begin{verbatim}
\cs_new_protected:Npn \@@_make_m_preamble_iv:nnn #1 #2 #3
{ \tl_gput_right:Nn \g_@@_preamble_tl
{ \@@_cell_begin:w
\begin { minipage } { \dim_eval:n { #3 } }
\mode_leave_vertical:
\arraybackslash
\vrule height \box_ht:N \@arstrutbox depth 0 pt width 0 pt
c \@@_cell_end:
}
\end{verbatim}
We test for the presence of a `<.

For $w$ and $W$

\begin{minipage}{\textwidth}
\begin{verbatim}
\@_cell_end:
\end{verbatim}
\end{minipage}

We test for the presence of a `<.

\begin{verbatim}
\@@_make_m_preamble_x:n
\end{verbatim}
\end{minipage}

After a specifier of column, we have to test whether there is one or several `<{..}.`
The command `\@@_put_box_in_flow_i:` is used when the value of `\l_@@_baseline_tl` is different
of `c` (which is the initial value and the most used).

Now, `\g_tampa_dim` contains the \textit{y}-value of the center of the array (the delimiters are centered in
relation with this value).

We take into account the position of the mathematical axis.

The following command is always used by \texttt{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).

With an environment \texttt{Matrix}, you want to remove the exterior \texttt{\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.
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}{\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:Nx \g_@@_aux_tl { \int_use:N \g_@@_notes_caption_int }
      \int_gzero:N \g_@@_notes_caption_int
    }
  }
  \tl_gput_right:Nx \g_@@_aux_tl { \exp_not:N \l_@@_note_in_caption_tl { \int_use:N \g_@@_notes_caption_int }
  }
  \int_gzero:N \g_@@_notes_caption_int
}\end{minipage}

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 now because there may be tabular notes in some blocks (which are not mono-column: the blocks which are mono-column have been composed in boxes yet)... and we have to create (potentially) the extra nodes before creating the blocks since there are medium nodes to create for the blocks.

$@@_create_extra_nodes:
$\seq_if_empty:NF \g_@@_notes_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 if compiles several 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:

$\cs_new_protected:Npn \@@_insert_caption:
The flag `\l_@@_in_caption_bool` affects only the behaviour of the command `\tabularnote` when used in the caption.

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.

In some circumstances (in particular when the package `caption` is loaded), the caption is composed several times. That’s why, when the same tabular note is encountered (in the caption!), we consider that you are in the second compilation and you can give to `\g_@@_notes_caption_int` its final value, which is the number of tabular notes in the caption. But sometimes, the caption is composed only once. In that case, we fix the value of `\g_@@_caption_finished_bool` now.

The TeX group is for potential specifications in the `\l_@@_notes_code_before_tl`.
We compose the tabular notes with a list of \texttt{enumitem}. The \texttt{\strut} and the \texttt{\unskip} are designed to give the ability to put a \texttt{\bottomrule} at the end of the notes with a good vertical space.

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

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

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

The case of \texttt{baseline} equal to \texttt{b}. Remember that, when the key \texttt{b} is used, the \texttt{\{array\}} (of \texttt{array}) is constructed with the option \texttt{t} (and not \texttt{b}). Now, we do the translation to take into account the option \texttt{b}.
Now, the general case.
\cs_new_protected:Npn \@@_use_arraybox_with_notes:
{
We convert a value of \textit{t} to a value of 1.
\tl_if_eq:NnT \l_@@_baseline_tl { t }
{ \cs_set_nopar:Npn \l_@@_baseline_tl { 1 } }
Now, we convert the value of \l_@@_baseline_tl (which should represent an integer) to an integer stored in \l_@_baseline_int.
\pgfpicture
\@@_qpoint:n { row - 1 }
\dim_gset_eq:NN \g_tmpa_dim \pgf@y
\str_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_@_baseline_int }
}
\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: } \}
}\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: } \}
\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: } \}
\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: } \}
\endpgfpicture
The command \texttt{\@\@@_put_box_in_flow_bis:} is used when the option \texttt{delimiters/max-width} is used because, in this case, we have to adjust the widths of the delimiters. The arguments \texttt{#1} and \texttt{#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.

\begin{verbatim}
\dim_zero_new:N \l_@@_real_left_delim_dim
\dim_zero_new:N \l_@@_real_right_delim_dim
\hbox_set:Nn \l_tmpb_box
{ \c_math_toggle_token \left #1 \vcenter
{ \vbox_to_ht:nn { \box_ht_plus_dp:N \l_tmpa_box } { } }
\right . \c_math_toggle_token }
\dim_set:Nn \l_@@_real_left_delim_dim { \box_wd:N \l_tmpb_box - \nulldelimiterspace }
\hbox_set:Nn \l_tmpb_box
{ \c_math_toggle_token \left . \vbox_to_ht:nn { \box_ht_plus_dp:N \l_tmpa_box } { } \right #2 \c_math_toggle_token }
\dim_set:Nn \l_@@_real_right_delim_dim { \box_wd:N \l_tmpb_box - \nulldelimiterspace }
\end{verbatim}

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

\begin{verbatim}
\skip_horizontal:N \l_@@_left_delim_dim
\skip_horizontal:N -\l_@@_real_left_delim_dim
\@@_put_box_in_flow:
\skip_horizontal:N \l_@@_right_delim_dim
\skip_horizontal:N -\l_@@_real_right_delim_dim
\end{verbatim}

The construction of the array in the environment \texttt{NiceArrayWithDelims} is, in fact, done by the environment \texttt{@@-light-syntax} or by the environment \texttt{@@-normal-syntax} (whether the option \texttt{light-syntax} is in force or not). When the key \texttt{light-syntax} is not used, the construction is a standard environment (and, thus, it's possible to use verbatim in the array).

\begin{verbatim}
\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 \texttt{\end} and, if it's the case, we test if this is the end of the environment (if it is not, an standard error will be raised by LaTeX for incorrect nested environments).
\end{verbatim}

\begin{verbatim}
\peek_remove_spaces:n
  { \peek_meaning:NTF \end \@@_analyze_end:Nn
    { \@@_transform_preamble: \\
      \@@_array: \g_@@_array_preamble_tl
    }
  }
\end{verbatim}

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

\begin{verbatim}
\exp_args:No \@@_array: \g_@@_array_preamble_tl
\end{verbatim}
When the key `light-syntax` is in force, we use an environment which takes its whole body as an argument (with the specifier `b`).

```latex
\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.

```latex
\tl_if_empty:nT { #1 } { \@@_fatal:n { empty~environment } }
\tl_map_inline:nn { #1 }
\str_if_eq:nnT { ##1 } { & } { \@@_fatal:n { ampersand~in~light-syntax } }
\str_if_eq: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 catched in the value of `\g_nicematrix_code_after_tl`. That doesn’t matter because `\CodeAfter` will be set to `no-op` before the execution of `\g_nicematrix_code_after_tl`.

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

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

```latex
\begin{array}{S}
```

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.

```latex
\@@_create_col_nodes:
\endarray
```

```latex
\cs_new_protected:Npn \@@_light_syntax_i:w #1 \CodeAfter #2 \q_stop
```

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

```latex
\seq_clear_new:N \l_@@_rows_seq
We rescan the character of end of line in order to have the correct catcode.
```

```latex
\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.

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

```latex
\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).

First, we treat the first row.

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

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.

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).

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.
The command \@@_create_col_nodes: will construct a special last row. That last row is a false row used to create the \texttt{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 \texttt{columns-width}).

\begin{verbatim}
\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 \pgfresetboundingbox
\str_if_empty:NF \l_@@_name_str
\{ \pgfnodealias \l_@@_name_str - col - 0 \}
\endpgfpicture
\skip_horizontal:N \g_@@_width_first_col_dim
\}
\omit \hbox
\{
\skip_horizontal:N -0.5\arrayrulewidth
\pgfsys@markposition \@@_env: - col - 1 \}
\skip_horizontal:N 0.5\arrayrulewidth
\}
\pgfpicture \pgfresetboundingbox
\str_if_empty:NF \l_@@_name_str
\{ \pgfnodealias \l_@@_name_str - col - 1 \}
\endpgfpicture
\}
\bool_gset_true:N \g_@@_row_of_col_done_bool
\endverbatim

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

\begin{verbatim}
\bool_gset_true:N \g_@@_row_of_col_done_bool
\end{verbatim}

First, we put a \texttt{col} node on the left of the first column (of course, we have to do that after the \omit).

\begin{verbatim}
\int_if_zero:nTF \l_@@_first_col_int
\{
\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 \pgfresetboundingbox
\str_if_empty:NF \l_@@_name_str
\{ \pgfnodealias \l_@@_name_str - 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 \pgfresetboundingbox
\str_if_empty:NF \l_@@_name_str
\{ \pgfnodealias \l_@@_name_str - col - 1 \}
\endpgfpicture
\}
\end{verbatim}

85
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 \texttt{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 \texttt{\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.

\begin{verbatim}
\skip_gset:Nn \g_tmpa_skip { 0 pt~plus 1 fill }
\bool_if:NTF \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
{ \bool_if:NT \l_@@_code_before_bool
  { \hbox
    { \skip_horizontal:N -0.5\arrayrulewidth
      \pgfsys@markposition { \@@_env: - col - 2 }
      \skip_horizontal:N 0.5\arrayrulewidth
    }
\pgfpicture
\pgfrememberpicturepositiononpagetrue
\pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
\str_if_empty:NF \l_@@_name_str
{ \pgfnodealias { \l_@@_name_str - col - 2 } { \@@_env: - col - 2 } }
\endpgfpicture
}\hbox}
\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 } \c_zero_int } }
{ \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 2 } \c_zero_int } }
\int_gincr:N \g_tmpa_int
\\texttt{\omit
\\texttt{\int_gincr:N \g_tmpa_int
\endverbatim}

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.

\begin{verbatim}
\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 } \c_zero_int } }
{ \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 2 } \c_zero_int } }
\\texttt{\omit
\\texttt{\int_gincr:N \g_tmpa_int
\endverbatim}

The incrementation of the counter $\g_tmpa_int$ must be done after the \texttt{\omit} of the cell.

\begin{verbatim}
\\texttt{\omit
\\texttt{\int_gincr:N \g_tmpa_int
\endverbatim}
We create the col node on the right of the current column.

```latex
\pgfpicture
\pgfrememberpicturepositiononpagefalse
\pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
{ \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
\str_if_empty:N \l_@@_name_str
{ \pgfnodealias
  { \l_@@_name_str - col - \int_eval:n { \g_tmpa_int + 1 } }
  { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
}
\endpgfpicture
\pgfpicture
\pgfrememberpicturepositiononpagefalse
\pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
{ \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
\str_if_empty:N \l_@@_name_str
{ \pgfnodealias
  { \l_@@_name_str - col - \int_eval:n { \g_tmpa_int + 1 } }
  { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
}
\endpgfpicture
\pgfpicture
\pgfrememberpicturepositiononpagefalse
\pgfcoordinate { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
{ \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
\str_if_empty:N \l_@@_name_str
{ \pgfnodealias
  { \l_@@_name_str - col - \int_eval:n { \g_tmpa_int + 1 } }
  { \@@_env: - col - \int_eval:n { \g_tmpa_int + 1 } }
}
\endpgfpicture
```

\&
\omit

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

```latex
\int_if_zero:nT \g_@@_col_total_int
{ \skip_gset:Nn \g_tmpa_skip { 0 pt~plus 1 fill } }
\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.

```latex
\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
\pgfrememberpicturepositiononpagefalse
\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 }
```

87
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 do 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:

The contents of the cell is constructed in the box \l_@@_cell_box because we have to compute some
dimensions of this box.
\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
{
Be careful: despite this letter 1 the cells of the “first column” are composed in a R manner since they are composed in a \bbox_overlap_left:n. 

\begin{verbatim}
\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 }
\end{verbatim}

We actualise the width of the “first column” because we will use this width after the construction of the array.

\begin{verbatim}
\dim_gset:Nn \g_@@_width_first_col_dim { \dim_max:nn \g_@@_width_first_col_dim { \box_wd:N \l_@@_cell_box } }
\end{verbatim}

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

\begin{verbatim}
\bbox_overlap_left:n
  { \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > \c_zero_dim
    { \@@_node_for_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
\end{verbatim}

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

\begin{verbatim}
\tl_const:Nn \c_@@_preamble_last_col_tl
  { > \bool_set_true:N \l_@@_in_last_col_bool
    \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
    \bool_gset_true:N \g_@@_last_col_found_bool
    \int_gincr:N \c@jCol
    \int_gset_eq:NN \g_@@_col_total_int \c@jCol
  }
\end{verbatim}

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

\begin{verbatim}
\cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
\end{verbatim}

With the flag \g_@@_last_col_found_bool, we will know that the “last column” is really used.

\begin{verbatim}
\bool_gset_true:N \g_@@_last_col_found_bool
\int_gincr:N \c@jCol
\int_gset_eq:NN \g_@@_col_total_int \c@jCol
\end{verbatim}

The contents of the cell is constructed in the box \l_tmpa_box because we have to compute some dimensions of this box.

\begin{verbatim}
\bbox_set:Nw \l_@@_cell_box \@@_math_toggle:
\@@_tuning_key_small:
\end{verbatim}
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 } { . }
  }
}

\begin{NiceArray}

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

\hbox_overlap_right:n

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

\NewDocumentEnvironment { NiceArray } { }

\begin{NiceArrayWithDelims} . .
\end{NiceArrayWithDelims}

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

\cs_new_protected:Npn \@@_def_env:nnn #1 #2 #3

\NewDocumentEnvironment { #1 NiceArray } { }

\begin{NiceArrayWithDelims} . .
\end{NiceArrayWithDelims}
The environment \{NiceMatrix\} and its variants

We define also an environment \{NiceMatrix\}
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 }

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

If the dimension \l_@@_width_dim is equal to 0 pt, that means that it has not be set by a previous use of \NiceMatrixOptions.

\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 }
16 After the construction of the array

The following command will be used when the key \texttt{rounded-corners} is in force (this is the key \texttt{rounded-corners} for the whole environment and not the key \texttt{rounded-corners} of a command \texttt{\Block}).
There was a \hook_gput_code:nnn { env / tabular / begin } { nicematrix } in the command \@@_pre_array_ii: in order to come back to the standard definition of \multicolumn (in the tabulars used by the final user in the cells of our array of nicematrix) and maybe another linked to colortbl.

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.

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.

It’s also time to give to \l_@@_last_row_int its real value.

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).

We need also the potential content of \g_@@_multicolumn_cells_seq.
Now, you create the diagonal nodes by using the \texttt{row} nodes and the \texttt{col} nodes.

\texttt{\@@_create_diag_nodes:}

We create the aliases using \texttt{last} for the nodes of the cells in the last row and the last column.

\begin{verbatim}
\begin{pgfpicture}
\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 }
}
\str_if_empty:NF \l_@@_name_str
{ \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 }
}
\endpgfpicture
\end{verbatim}

By default, the diagonal lines will be parallelized\textsuperscript{11}. There are two types of diagonals lines: the $\Ddots$ diagonals and the $\Iddots$ diagonals. We have to count both types in order to know whether a diagonal is the first of its type in the current \texttt{NiceArray} environment.

\texttt{bool_if:NT \l_@@_parallelize_diags_bool}

\begin{verbatim}
\int_gzero_new:N \g_@@_ddots_int
\int_gzero_new:N \g_@@_iddots_int
\dim_gzero_new:N \g_@@_delta_x_one_dim
\dim_gzero_new:N \g_@@_delta_y_one_dim
\dim_gzero_new:N \g_@@_delta_x_two_dim
\dim_gzero_new:N \g_@@_delta_y_two_dim
\int_zero_new:N \l_@@_initial_i_int
\int_zero_new:N \l_@@_initial_j_int
\end{verbatim}

\textsuperscript{11}It’s possible to use the option \texttt{parallelize-diags} to disable this parallelization.
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 {
    \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_end_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 }

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_seq which will contain all the cells which are empty (and not in a block) considered in the corners of the array.

\@@_compute_corners:

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:
\tl_if_empty:NF \l_@@_hlines_clist \@@_draw_hlines:
\tl_if_empty:NF \l_@@_vlines_clist \@@_draw_vlines:

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

\IfPackageLoadedTF { tikz }
{
    \tikzset
    {
        every-picture /.style =
        {
            overlay ,
            remember-picture ,
            name-prefix = \@@_env: -
        }
    }
}
\bool_if:NT \c_@@_tagging_array_bool {
    \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
\g_@@_pre_code_after_tl
\tl_gclear:N \g_@@_pre_code_after_tl
When \texttt{light-syntax} is used, we insert systematically a \texttt{\CodeAfter} in the flow. Thus, it’s possible to have two instructions \texttt{\CodeAfter} and the second may be in \texttt{\g_nicematrix_code_after_tl}. That’s why we set \texttt{\CodeAfter} to be \texttt{no-op} now.

\texttt{\cs_set_eq:NN \CodeAfter \prg_do_nothing:}

We clear the list of the names of the potential \texttt{\SubMatrix} that will appear in the \texttt{\CodeAfter} (unfortunately, that list has to be global).

\texttt{\seq_gclear:N \g_@@_submatrix_names_seq}

The following code is a security for the case the user has used \texttt{babel} with the option \texttt{spanish}: in that case, the characters > and < are activated and Tikz is not able to solve the problem (even with the Tikz library \texttt{babel}).

\texttt{\% \int_compare:nNnT { \char_value_catcode:n { 60 } } = { 13 } \%
\{ \@@_rescan_for_spanish:N \g_nicematrix_code_after_tl \}}

And here’s the \texttt{\CodeAfter}. Since the \texttt{\CodeAfter} may begin with an “argument” between square brackets of the options, we extract and treat that potential “argument” with the command \texttt{\@@_CodeAfter_keys:}.

\texttt{\bool_set_true:N \l_@@_in_code_after_bool}

\texttt{\exp_last_unbraced:N \@@_CodeAfter_keys: \g_nicematrix_code_after_tl}

\texttt{\scan_stop:}

\texttt{\tl_gclear:N \g_nicematrix_code_after_tl}

\texttt{\group_end:}

\texttt{\g_@@_pre_code_before_tl} is for instructions in the cells of the array such as \texttt{\rowcolor} and \texttt{\cellcolor} (when the key \texttt{color-inside} is in force). These instructions will be written on the \texttt{aux} file to be added to the \texttt{code-before} in the next run.

\texttt{\seq_if_empty:NF \g_@@_rowlistcolors_seq \{ \@@_clear_rowlistcolors_seq: \}}

\texttt{\tl_if_empty:NF \g_@@_pre_code_before_tl}

\texttt{\tl_gput_right:Nx \g_@@_aux_tl}

\texttt{\tl_gset:Nn \exp_not:N \g_@@_pre_code_before_tl}

\texttt{\exp_not:o \g_@@_pre_code_before_tl}

\texttt{\tl_gclear:N \g_@@_pre_code_before_tl}

\texttt{\tl_if_empty:NF \g_nicematrix_code_before_tl}

\texttt{\tl_gput_right:Nx \g_@@_aux_tl}

\texttt{\tl_gset:Nn \exp_not:N \g_@@_code_before_tl}

\texttt{\exp_not:o \g_nicematrix_code_before_tl}

\texttt{\tl_gclear:N \g_nicematrix_code_before_tl}

\texttt{\str_gclear:N \g_@@_name_env_str}

\texttt{\@@_restore_iRow_jCol:}

The command \texttt{\CT@arc@} contains the instruction of color for the rules of the array\textsuperscript{12}. This command is used by \texttt{\CT@arc@} but we use it also for compatibility with \texttt{colortbl}. But we want also to be able to use color for the rules of the array when \texttt{colortbl} is \texttt{not} loaded. That’s why we do the following instruction which is in the patch of the end of arrays done by \texttt{colortbl}.

\texttt{\cs_gset_eq:NN \CT@arc@ \@@_old_CT@arc@}

\texttt{\}

\textsuperscript{12}e.g. \texttt{\color[rgb]{0.5,0.5,0}}
The following command will extract the potential options (between square brackets) at the beginning of the `\CodeAfter` (that is to say, when `\CodeAfter` is used, the options of that “command” `\CodeAfter`). Idem for the `\CodeBefore`.

```latex
\NewDocumentCommand \@@_CodeAfter_keys: { O { } } \{ \keys_set:nn { NiceMatrix / CodeAfter } { #1 } \}
```

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

```latex
\cs_new_protected:Npn \@@_adjust_pos_of_blocks_seq:
\{ \seq_gset_map_x:NNn \g@@pos_of_blocks_seq \g@@pos_of_blocks_seq \{ \@@_adjust_pos_of_blocks_seq_i:nnnnn ##1 \}
\}
```

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).

```latex
\hook_gput_code:nnn { begindocument } { . } \{ \cs_new_protected:Npx \@@draw_dotted_lines: \{ \c@@pgfortikzpicture_tl \@@draw_dotted_lines_i: \c@@endpgfortikzpicture_tl \}
\}
```

The following command must not be protected.

```latex
\cs_new:Npn \@@_adjust_pos_of_blocks_seq_i:nnnnn #1 #2 #3 #4 #5
\{ \#1 \}
\{ \#2 \}
\{ \int_compare:nNnTF { #3 } > { 99 } \{ \int_use:N \c@iRow \}
\{ #3 \}
\}
\{ \int_compare:nNnTF { #4 } > { 99 } \{ \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).

```latex
\hook_gput_code:nnn { begindocument } { . } \{ \cs_new_protected:Npx \@@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 \}
\}
```

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

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

```latex
\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 \pgf@x = 0.2 \pgf@x \pgf@y = 0.2 \pgf@y }
  \anchor { center } { \pgfpointorigin }
  \anchor { 1 } { \five \pgf@x = 0.4 \pgf@x \pgf@y = 0.4 \pgf@y }
  \anchor { 2 } { \five \pgf@x = 0.6 \pgf@x \pgf@y = 0.6 \pgf@y }
  \anchor { 3 } { \five \pgf@x = 0.8 \pgf@x \pgf@y = 0.8 \pgf@y }
  \anchor { 4 } { \five \pgf@x = 1.0 \pgf@x \pgf@y = 1.0 \pgf@y }
  \anchor { 6 } { \five \pgf@x = 1.4 \pgf@x \pgf@y = 1.4 \pgf@y }
  \anchor { 7 } { \five \pgf@x = 1.6 \pgf@x \pgf@y = 1.6 \pgf@y }
  \anchor { 8 } { \five \pgf@x = 1.8 \pgf@x \pgf@y = 1.8 \pgf@y }
  \anchor { 9 } { \five \pgf@x = 2.0 \pgf@x \pgf@y = 2.0 \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).

```latex
\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_gset_eq:NN \pgf@x \l_tma_dim \dim_gset_eq:NN \pgf@y \l_tmb_dim
    \@@_qpoint:n { row - \int_min:nn { ##1 } { \c@iRow + 1 } } \dim_gset_eq:NN \pgf@x \l_tmc_dim \dim_gset_eq:NN \pgf@y \l_tmd_dim
    \@@_qpoint:n { col - \int_min:nn { ##1 + 1 } { \c@jCol + 1 } } \dim_gset_eq:NN \pgf@x \l_tmcp_dim \dim_gset_eq:NN \pgf@y \l_tmmd_dim
    \@@_qpoint:n { row - \int_min:nn { ##1 + 1 } { \c@iRow + 1 } } \dim_gset_eq:NN \pgf@x \l_tmpp_dim \dim_gset_eq:NN \pgf@y \l_tmmb_dim
    \pgfnode { \@@_diag_node } { center } { } { \@@_env: - ##1 } { }
    \str_if_empty:NF \l_@@_name_str { \pgfnodealias { \l_@@_name_str - ##1 } { \@@_env: - ##1 } }
  }
  \pgftransformshift { \pgfpoint \l_tma_dim \l_tmb_dim }
}
```

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

```latex
\int_step_inline:nn { \int_max:nn \c@iRow \c@jCol + 1 }
{ \@@_qpoint:n { col - \int_min:nn { #1 } { \c@jCol + 1 } } \dim_gset_eq:NN \l_tma_dim \pgf@x \dim_gset_eq:NN \l_tmb_dim \pgf@y
  \dim_gset_eq:NN \l_tmc_dim \pgf@x \dim_gset_eq:NN \l_tmmd_dim \pgf@y
  \dim_gset_eq:NN \l_tmpp_dim \pgf@x \dim_gset_eq:NN \l_tmmb_dim \pgf@y
  \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.

```latex
\int_step_inline:nn { \int_max:nn \c@iRow \c@jCol + 1 }
{ \@@_qpoint:n { row - \int_min:nn { \l_tma_dim } { \c@iRow + 1 } } \dim_gset_eq:NN \l_tmb_dim \pgf@y
  \dim_gset_eq:NN \l_tmmd_dim \pgf@y
  \dim_gset_eq:NN \l_tmpp_dim \pgf@y
  \pgfnode { \@@_diag_node } { center } { } { \@@_env: - \l_tmb_dim } { }
  \pgfnodealias
  { \l_@@_name_str - #1 } { \@@_env: - \l_tmb_dim }
}
```

99
17 We draw the dotted lines

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

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

The command \texttt{\@@_find_extremities_of_line:nnnn} takes four arguments:

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

This command computes:

- \texttt{\l@@_initial\_i\_int} and \texttt{\l@@_initial\_j\_int} which are the coordinates of one extremity of the line;
- \texttt{\l@@_final\_i\_int} and \texttt{\l@@_final\_j\_int} which are the coordinates of the other extremity of the line;
- \texttt{\l@@_initial\_open\_bool} and \texttt{\l@@_final\_open\_bool} to indicate whether the extremities are open or not.

```
cs_new_protected:Npn \@@_find_extremities_of_line:nnnn #1 #2 #3 #4
does something.
```

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

```
cs_set:cpn { @@_dotted_#1 - #2 }
```

Initialization of variables.

```
\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 }
```
We will do two loops: one when determinating the initial cell and the other when determinating the final cell. The boolean \texttt{\_\_\_\_\_\_stop_loop_bool} will be used to control these loops. In the first loop, we search the “final” extremity of the line.

\begin{verbatim}
\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 }
  \int_compare:nNnT \l_\_\_\_final_i_int > \l_\_\_\_row_max_int {
    \int_compare:nNnT \int_use:N \l_\_\_\_final_j_int > \l_\_\_\_col_max_int {
      \bool_set_true:N \l_\_\_\_final_open_bool }
    } \int_compare:nNnT \l_\_\_\_final_j_int < \l_\_\_\_col_min_int {
    \int_compare:nNnT \int_use:N \l_\_\_\_final_j_int > \l_\_\_\_col_max_int {
      \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 }
  }
\end{verbatim}

We test if we are still in the matrix.

\begin{verbatim}
\bool_set_false:N \l_\_\_\_\_final_open_bool
\int_compare:nNnTF \l_\_\_\_final_i_int > \l_\_\_\_row_max_int {
  \int_compare:nNnT \int_use:N \l_\_\_\_final_i_int = \c_one_int {
    \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 \int_use:N \l_\_\_\_final_j_int = \c_one_int {
    \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 }
}
\bool_if:NTF \l_\_\_\_final_open_bool
\end{verbatim}

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.

\begin{verbatim}
\int_sub:Nn \l_\_\_\_final_i_int { #3 }
\int_sub:Nn \l_\_\_\_final_j_int { #4 }
\bool_set_true:N \l_\_\_\_stop_loop_bool
\end{verbatim}

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 \texttt{\_\_\_\_\_\_final_i_int} and \texttt{\_\_\_\_\_\_final_j_int}.

\begin{verbatim}
\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
}
\end{verbatim}
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.

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

```
\bool_set:cn \l_@@_final_i_int \cs_set:cpn
{ \l_@@_dotted
  \int_use:N \l_@@_final_i_int
  \int_use:N \l_@@_final_j_int
  \endcs_set:cpn

\bool_set_true:N \l_@@_stop_loop_bool
\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 } = \c_one_int
      \bool_set_true:N \l_@@_initial_open_bool
    \int_compare:nNnT \l_@@_initial_j_int = { \l_@@_col_min_int - 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
```
\begin{verbatim}
\@_ 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:cpn
{ \@_ dotted_
\int_use:N \l_@@_initial_i_int - 
\int_use:N \l_@@_initial_j_int }
{ }
}
\}
\}
\end{verbatim}

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:Nx \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 \}
\{ \} % for the name of the block

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

\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 sub-matrices which contains the cell of row \#1 and column \#2. As of now, it’s only the whole array (excepted exterior rows and columns).
We do a loop over all the submatrices specified in the code-before. We have stored the position of all those submatrices in \texttt{\g@@submatrix_seq}.

\begin{verbatim}
\seq_map_inline:Nn \g@@submatrix_seq
{ \@@_adjust_to_submatrix:nnnnnn { #1 } { #2 } ##1 }
\end{verbatim}

\#1 and \#2 are the numbers of row and columns of the cell where the command of dotted line (ex.: \texttt{\Vdots}) has been issued. \#3, \#4, \#5 and \#6 are the specification (in \texttt{i} and \texttt{j}) of the submatrix we are analyzing.

\begin{verbatim}
\cs_set_protected:Npn \@@_adjust_to_submatrix:nnnnnn #1 #2 #3 #4 #5 #6
{ \int_compare:nNnF { #3 } > { #1 } { \int_compare:nNnF { #1 } > { #5 } { \int_compare:nNnF { #4 } > { #2 } { \int_compare:nNnF { #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 } \} } } } \int_set:Nn \l@@row_min_int \{ \int_max:nn \l@@row_min_int { \l_@@row_min_int } \} \int_set:Nn \l@@col_min_int \{ \int_max:nn \l@@col_min_int { \l_@@col_min_int } \} \int_set:Nn \l@@row_max_int \{ \int_min:nn \l@@row_max_int { \l_@@row_max_int } \} \int_set:Nn \l@@col_max_int \{ \int_min:nn \l@@col_max_int { \l_@@col_max_int } \} } \int_set:Nn \l_@@row_min_int 1 \int_set:Nn \l_@@col_min_int 1 \int_set_eq:NN \l_@@row_max_int \c@iRow \int_set_eq:NN \l_@@col_max_int \c@jCol
\end{verbatim}
If, in fact, all the cells of the column are empty (no PGF/Tikz nodes in those cells).

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

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.
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.

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.

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.

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.
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 (?).

\begin{Verbatim}
\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
\end{Verbatim}

\@@_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.

\begin{Verbatim}
\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_of_line:nnnn { #1 } { #2 } 0 1
}{
\group_begin:
\@@_open_shorten:
\int_if_zero:nTF { #1 } { \color { nicematrix-first-row } }
{ \color { nicematrix-last-row } }
\keys_set:nn { NiceMatrix / xdots } { #3 }
\tl_if_empty:oF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
\@@_actually_draw_Cdots:
\group_end:
}
\end{Verbatim}

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.

\begin{Verbatim}
\group_begin:
\@@_open_shorten:
\int_if_zero:nTF { #1 } { \color { nicematrix-first-row } }
{ \color { nicematrix-last-row } }
{ \keys_set:nn { NiceMatrix / xdots } { #3 } }
\tl_if_empty:oF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
\@@_actually_draw_Cdots:
\group_end:
\end{Verbatim}

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.

\begin{Verbatim}
\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 } \}
\end{Verbatim}
\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_set:Nn \l_\@_y_initial_dim \\
{ \dim_max: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_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 \\
{ \dim_min:nn \l_\@_y_final_dim \pgf@y } }
}
\dim_compare:nNnT \l_\@_y_final_dim = \c_max_dim
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_of_line:nnnn \{ \#1 \} \{ \#2 \} 1 0 \}
The previous command may have changed the current environment by marking some cells as “dotted”, but, fortunately, it is outside the group for the options of the line.

\cs_new_protected:Npn \@@_actually_draw_Vdots: \{}
\begin{group}
\@@_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 \}
\tl_if_empty:oF \l_@@_xdots_color_tl 
{ \color { \l_@@_xdots_color_tl } }
\@@_actually_draw_Vdots: 
\end{group}

The command \@@_actually_draw_Vdots: has the following implicit arguments:

- \l_@@_initial_i_int
- \l_@@_initial_j_int
- \l_@@_initial_open_bool
- \l_@@_final_i_int
- \l_@@_final_j_int
- \l_@@_final_open_bool.

The following function is also used by \Vdotsfor.

\cs_new_protected:Npn \@@_actually_draw_Vdots: \{}
\begin{group}
\@@_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 \}
\tl_if_empty:oF \l_@@_xdots_color_tl 
{ \color { \l_@@_xdots_color_tl } }
\@@_actually_draw_Vdots: 
\end{group}

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

\bool_lazy_and:nTF \l_@@_initial_open_bool \l_@@_final_open_bool
\We have to determine the \( x \)-value of the vertical rule that we will have to draw.

\begin{group}
\@@_open_y_initial_dim: 
\@@_open_y_final_dim: 
\int_if_zero:nTF \{ \#2 \} \l_@@_initial_i_int
\end{group}

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

\begin{group}
\@@_qpoint:n \{ \col - 1 \}
\dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
\dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim 
\dim_sub:Nn \l_@@_x_initial_dim \l_@@_extra_left_margin_dim
\end{group}
We have a dotted line open on both sides in the “last column”.

\[\text{\dim_set_eq:NN } \l_@@_x_initial_dim \pgf@x\]
\[\text{\dim_add:Nn } \l_@@_x_initial_dim \l_@@_right_margin_dim\]
\[\text{\dim_add:Nn } \l_@@_x_initial_dim \l_@@_extra_right_margin_dim\]
\[\text{\dim_add:Nn } \l_@@_x_initial_dim \c_@@_shift_exterior_Vdots_dim\]

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

\[\text{\dim_set_eq:NN } \l_@@_x_initial_dim { ( \pgf@x + \l_tmpa_dim ) / 2 }\]

Now, the dotted line is not open on both sides (maybe open on only one side).

The boolean \l_\tmpa Bool indicates whether the column is of type \texttt{l} or may be considered as if.

\[\text{\bool_set_false:N } \l_\tmpa Bool\]
\[\text{\bool_if:NF } \l_@@_initial_open_bool\]
\[\text{\bool_if:NF } \l_@@_final_open_bool\]
\[\text{\@@_set_initial_coords_from_anchor:n } \text{south-west}\]
\[\text{\@@_set_final_coords_from_anchor:n } \text{north-west}\]
\[\text{\dim_set:Nn } \l_\tmpa dim \l_@@_x_initial_dim = \l_@@_x_final_dim\]

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

\[\text{\bool_if:NTF } \l_@@_initial_open_bool\]
\[\text{\@@_open_y_initial_dim:}\]
\[\text{\@@_set_final_coords_from_anchor:n } \text{north}\]
\[\text{\dim_set_eq:NN } \l_@@_x_initial_dim \l_@@_x_final_dim\]

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

\[\text{\bool_if:NTF } \l_@@_initial_open_bool\]
\[\text{\@@_open_y_initial_dim:}\]
\[\text{\@@_set_final_coords_from_anchor:n } \text{north}\]
\[\text{\dim_compare_p:nNn } \l_@@_x_initial_dim = \l_@@_x_final_dim\]

Now the case where both extremities are closed. The first conditional tests whether the column is of type \texttt{c} or may be considered as if.

\[\text{\bool_if:NTF } \l_\tmpa Bool \text{\dim_min:nn } \text{\dim_max:nn}\]
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.

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.

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.

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.
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_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_of_line:nnnn { #1 } { #2 } 1 { -1 }
The previous command may have changed the current environment by marking some cells as “dotted”,
but, fortunately, it is outside the group for the options of the line.
\group_begin:
\@@_open_shorten:
\keys_set:nn { NiceMatrix / xdots } { #3 }
\tl_if_empty:oF \l@@_xdots_color_tl { \color { \l@@_xdots_color_tl }
\@@_actually_draw_Iddots:
\group_end:
\@@_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.
The actual instructions for drawing the dotted lines with Tikz

The command \texttt{\@\_draw\_line:} should be used in a \texttt{\{pgfpicture\}). It has six implicit arguments:

- \texttt{\l\_\@\_x\_initial\_dim}
- \texttt{\l\_\@\_y\_initial\_dim}
- \texttt{\l\_\@\_x\_final\_dim}
- \texttt{\l\_\@\_y\_final\_dim}
- \texttt{\l\_\@\_initial\_open\_bool}
- \texttt{\l\_\@\_final\_open\_bool}

\begin{verbatim}
18 The actual instructions for drawing the dotted lines with Tikz
The command \texttt{\@\_draw\_line:} should be used in a \texttt{\{pgfpicture\}). It has six implicit arguments:
\begin{itemize}
  \item \texttt{\l\_\@\_x\_initial\_dim}
  \item \texttt{\l\_\@\_y\_initial\_dim}
  \item \texttt{\l\_\@\_x\_final\_dim}
  \item \texttt{\l\_\@\_y\_final\_dim}
  \item \texttt{\l\_\@\_initial\_open\_bool}
  \item \texttt{\l\_\@\_final\_open\_bool}
\end{itemize}
\end{verbatim}
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.

```latex
\cs_new_protected:Npn \@@_draw_unstandard_dotted_line:n #1
\begin {scope}
\@@_draw_unstandard_dotted_line:o { \l_@@_xdots_line_style_tl , \l_@@_xdots_color_tl }
\end {scope}
```

We have used the fact that, in PGF, un color name can be put directly in a list of options (that’s why we have put directly `\l_@@_xdots_color_tl`).

The argument of `\@@_draw_unstandard_dotted_line:n` is, in fact, the list of options.

```latex
\cs_new_protected:Npn \@@_draw_unstandard_dotted_line:nnnn #1 #2 #3 #4
```

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

```latex
\hook_gput_code:nnn { begindocument } { . }
\IfPackageLoadedTF { tikz }
\tikzset
\@@_node_above / .style = { sloped , above },
\@@_node_below / .style = { sloped , below },
\@@_node_middle / .style =
\{ sloped ,
inner_sep = \c_@@_innersep_middle_dim
\}
\hook_gput_code:nnn { enddocument } { . }
```

We take into account the parameters `xdots/shorten-start` and `xdots/shorten-end` "by hand" because, when we use the key `shorten` > and `shorten <` of TikZ in the command `\draw`, we don’t have the expected output with `{decorate,decoration=brace}` is used.

The dimension `\l_@@_l_dim` is the length ℓ of the line to draw. We use the floating point reals of the L3 programming layer to compute this length.
It seems that, during the first compilations, the value of $l_{\text{\tiny final\_dim}}$ 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.

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

Be careful: We can’t put $\c_{\text{\tiny math\_toggle\_token}}$ instead of $\$ in the following lines because we are in the contents of Tikz nodes (and they will be rescanned if the Tikz library babel is loaded).

```latex
\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 }
\end { scope }
```
The command `\@@_draw_standard_dotted_line:` draws the line with our system of dots (which gives a dotted line with real rounded dots).

The dimension \l@@_l_dim is the length ℓ of the line to draw. We use the floating point reals of the L3 programming layer to compute this length.

```
\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:
```

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_end_dim \l@@_l_dim }
\cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:nnnn { n o o o }
```

The command `\@@_draw_standard_dotted_line:` draws the line with our system of dots (which gives a dotted line with real rounded dots).

```latex
\cs_new_protected:Npn \@@_draw_standard_dotted_line:
{ 
\group_begin:

The dimension \l@@_l_dim is the length ℓ of the line to draw. We use the floating point reals of the L3 programming layer to compute this length.

```
\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:
```

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_end_dim \l@@_l_dim }
\cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:nnnn { n o o o }
```

The command `\@@_draw_standard_dotted_line:` draws the line with our system of dots (which gives a dotted line with real rounded dots).

```latex
\cs_new_protected:Npn \@@_draw_standard_dotted_line:
{ 
\group_begin:

The dimension \l@@_l_dim is the length ℓ of the line to draw. We use the floating point reals of the L3 programming layer to compute this length.

```
\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:
```

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_end_dim \l@@_l_dim }
\cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:nnnn { n o o o }
```

The command `\@@_draw_standard_dotted_line:` draws the line with our system of dots (which gives a dotted line with real rounded dots).

```latex
\cs_new_protected:Npn \@@_draw_standard_dotted_line:
{ 
\group_begin:

The dimension \l@@_l_dim is the length ℓ of the line to draw. We use the floating point reals of the L3 programming layer to compute this length.

```
\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:
```

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_end_dim \l@@_l_dim }
\cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:nnnn { n o o o }
```

The command `\@@_draw_standard_dotted_line:` draws the line with our system of dots (which gives a dotted line with real rounded dots).

```latex
\cs_new_protected:Npn \@@_draw_standard_dotted_line:
{ 
\group_begin:

The dimension \l@@_l_dim is the length ℓ of the line to draw. We use the floating point reals of the L3 programming layer to compute this length.

```
\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:
```

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_end_dim \l@@_l_dim }
\cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:nnnn { n o o o }
```
The dimensions $\l_{\mathrm{tmpa}} \dim$ and $\l_{\mathrm{tmpb}} \dim$ are the coordinates of the vector between two dots in the dotted line.

\begin{verbatim}
\dim_set:Nn \l_{\mathrm{tmpa}} \dim \\
\{ \dim_ratio:nn \l_{\mathrm{xx}} \dim \l_{\mathrm{xdots_inter}} \dim \l_{\mathrm{xl}} \dim \}
\dim_set:Nn \l_{\mathrm{tmpb}} \dim \\
\{ \dim_ratio:nn \l_{\mathrm{xx}} \dim \l_{\mathrm{xdots_inter}} \dim \l_{\mathrm{xl}} \dim \}
\end{verbatim}

In the loop over the dots, the dimensions $\l_{\mathrm{xx}} \dim_{\mathrm{x_initial}} \dim$ and $\l_{\mathrm{yy}} \dim_{\mathrm{y_initial}} \dim$ will be used for the coordinates of the dots. But, before the loop, we must move until the first dot.

\begin{verbatim}
\dim_gadd:Nn \l_{\mathrm{xx}} \dim_{\mathrm{x_initial}} \dim \\
\{ ( \dim_ratio:nn \l_{\mathrm{xx}} \dim \l_{\mathrm{xdots_inter}} \dim \l_{\mathrm{xl}} \dim ) * \l_{\mathrm{tmpa}} \dim \}
\dim_gadd:Nn \l_{\mathrm{yy}} \dim_{\mathrm{y_initial}} \dim \\
\{ ( \dim_ratio:nn \l_{\mathrm{xx}} \dim \l_{\mathrm{xdots_inter}} \dim \l_{\mathrm{xl}} \dim ) * \l_{\mathrm{tmpa}} \dim \}
\end{verbatim}
\l_00_y_final_dim - \l_00_y_initial_dim,
\l_00_x_final_dim - \l_00_x_initial_dim
\pgftransformrotate { \fp_use:N \l_tmpa_fp }
\bool_if:NF \l_@@_xdots_h_labels_bool { \fp_zero:N \l_tmpa_fp }
\tl_if_empty:NF \l_@@_xdots_middle_tl
{\begin { pgfscope }
\pgfset { inner~sep = \c_@@_innersep_middle_dim }
\pgfnode
{ rectangle }
{ center }
{ \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
{ \c_math_toggle_token
\scriptstyle \l_@@_xdots_middle_tl
\c_math_toggle_token
}
}
{ \pgfusepath { fill }
}
\end { pgfscope }
}
\tl_if_empty:NF \l_@@_xdots_up_tl
{\pgfnode
{ rectangle }
{ south }
{ \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
{ \c_math_toggle_token
\scriptstyle \l_@@_xdots_up_tl
\c_math_toggle_token
}
}
{ \pgfusepath { } }
}
\tl_if_empty:NF \l_@@_xdots_down_tl
{\pgfnode
{ rectangle }
{ north }
{ \rotatebox { \fp_eval:n { - \l_tmpa_fp } }
{ \c_math_toggle_token
\scriptstyle \l_@@_xdots_down_tl
\c_math_toggle_token
}
}
{ \pgfusepath { } }
}
\endpgfscope
19 User commands available in the new environments

The commands \@@_Ldots, \@@_Cdots, \@@_Vdots, \@@_Ddots and \@@_Iddots will be linked to \Ldots, \Cdots, \Vdots, \Ddots and \Iddots in the environments \{NiceArray\} (the other environments of nicematrix rely upon \{NiceArray\}).

The syntax of these commands uses the character _ as embellishment and thats’ why we have to insert a character _ in the \textit{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 \textit{arg spec} will be rescanned.

\begin{verbatim}
\hook_gput_code:nnn { begindocument } { . }
\{
\cs_set_nopar:Npn \l_@@_argspec_tl { m E { _ ^ : } { { } { } { } } }
\tl_set_rescan:Nno \l_@@_argspec_tl \l_@@_argspec_tl
\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_if_zero:nTF \c@iRow
{ \@@_error:nn { in~first~row } \Ddots }
{ \int_compare:nNnTF \c@iRow = \l_@@_last_row_int
{ \@@_error:nn { in~last~row } \Ddots }
{ \@@_instruction_of_type:nnn \c_false_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_if_zero:nTF \c@iRow
{ \@@_error:nn { in~first~row } \Iddots }
{ \int_compare:nNnTF \c@iRow = \l_@@_last_row_int
{ \@@_error:nn { in~last~row } \Iddots }
{ \@@_instruction_of_type:nnn \c_false_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{verbatim}
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_set_eq:NN \@@_old_multicolumn \multicolumn

The command \@@_Hdotsfor will be linked to \Hdotsfor in \{NiceArrayWithDelims\}. Tikz nodes are created also in the implicit cells of the \Hdotsfor (maybe we should modify that point).

This command must not be protected since it begins with \multicolumn.
\cs_new:Npn \@@_Hdotsfor:
\{ 
\bool_lazy_and:nTF \{ \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 } { . } 
\cs_set_nopar:Npn \l_@@_argspec_tl { \m m O { } E { _ ^ : } { } { } } 
\tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl

We don’t put ! before the last optional argument for homogeneity with \Cdots, etc. which have only one optional argument.
\texttt{tl_gput_right:Nx \v@HVdotsfor_lines_tl}

\texttt{\textbackslash@Hdotsfor:nnnn}

\texttt{\\int\_use:N \c@iRow}
\texttt{\\int\_use:N \c@jCol}
\texttt{\#2}

\texttt{
\#1 , \#3 ,
\textup{down} = \\exp\_not:n \{ \#4 \},
\textup{up} = \\exp\_not:n \{ \#5 \},
\textup{middle} = \\exp\_not:n \{ \#6 \}
}

\texttt{\textbackslash prg\_replicate:nn \{ \#2 - 1 \}}

&
\texttt{\\multicolumn\{1\}\{c\}\{}}
\texttt{\textbackslash cs\_set\_eq:NN \textbackslash Code\_After \textbackslash\textbackslash@Code\_After_i:}

\texttt{\textbackslash cs\_new\_protected:Npn \textbackslash@Hdotsfor:nnnn \#1 \#2 \#3 \#4}

\texttt{\textbackslash boolean\_set\_false:N \l_\textbackslash@initial\_i_int}
\texttt{\textbackslash boolean\_set\_false:N \l_\textbackslash@final\_i_int}

\textbf{For the row, it’s easy.}
\texttt{\\int\_set:Nn \l_\textbackslash@initial\_i_int \{ \#1 \}}
\texttt{\\int\_set\_eq:NN \l_\textbackslash@final\_i_int \l_\textbackslash@initial\_i_int}

\textbf{For the column, it’s a bit more complicated.}
\texttt{\\int\_compare:nNnTF \{ \#2 \} = \textbackslash c\_one\_int}

\texttt{\\int\_set\_eq:NN \l_\textbackslash@initial\_j_int \textbackslash c\_one\_int}

\texttt{\\boolean\_set\_true:N \l_\textbackslash@initial\_open\_bool}

\{\texttt{\textbackslash cs\_if\_exist:cTF}

\{\texttt{\pgf\textbackslash sh\textbackslash ns\textbackslash \textbackslash\textbackslash env:}
\texttt{\\int\_use:N \l_\textbackslash@initial\_i_int}
\texttt{\\int\_eval:n \{ \#2 - 1 \}}
\}

\{\texttt{\\int\_set:Nn \l_\textbackslash@initial\_j_int \{ \#2 - 1 \}}
\}

\{\texttt{\\int\_set:Nn \l_\textbackslash@initial\_j_int \{ \#2 \}}
\}

\{\texttt{\\boolean\_set\_true:N \l_\textbackslash@initial\_open\_bool}
\}

\}

\texttt{\\int\_compare:nNnTF \{ \#2 + \#3 -1 \} = \textbackslash c\@jCol}

\{\texttt{\\int\_set:Nn \l_\textbackslash@final\_j_int \{ \#2 + \#3 - 1 \}}
\}

\{\texttt{\\boolean\_set\_true:N \l_\textbackslash@final\_open\_bool}
\}

\}

\{\texttt{\textbackslash cs\_if\_exist:cTF}

\{\texttt{\pgf\textbackslash sh\textbackslash ns\textbackslash \textbackslash\textbackslash env:}
\texttt{\\int\_use:N \l_\textbackslash@final\_i_int}
\texttt{\\int\_eval:n \{ \#2 + \#3 \}}
\}

\{\texttt{\\int\_set:Nn \l_\textbackslash@final\_j_int \{ \#2 + \#3 \}}
\}

\}

122
We declare all the cells concerned by the \Hdotsfor as “dotted” (for the dotted lines created by \Cdots, \dots, etc., this job is done by \@@_find_extremeities_of_line:nmmn). This declaration is done by defining a special control sequence (to nil).

\int_step_inline:nnn { #2 } { #2 + #3 - 1 }
\{ \cs_set:cpn { \_dotted \_#1 - ##1 } { } \}
\hook_gput_code:nnn { begindocument } { . }
\{ \cs_set_nopar:Npn \_\_\argspec_tl { m m O { } E { _ ^ : } { { } { } { } } } \tl_set_rescan:Nno \_\_\argspec_tl { } \_\_\argspec_tl \cs_new_protected:Npn \@@_Vdotsfor:
{ \@@_collect_options:n { \@@_Vdotsfor_i } }
\exp_args:NNo \NewDocumentCommand \@@_Vdotsfor_i \l_\_\argspec_tl
{ \bool_set_false:N \l_\_initial_open_bool
\bool_set_false:N \l_\_final_open_bool
\tl_gput_right:Nx \_\_HVdotsfor_lines_tl
{ \@@_Vdotsfor:nnnn }
\tl_gput_right:Nx \_\_\HVdotsfor_lines_tl
{ \_\_dotted \_#1 - \#1 } } }
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
}
\}
\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 }
\tl_if_empty:oF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
\l_@@_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_of_line:nmmn). This declaration is done by defining a special control sequence (to nil).

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

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

\NewDocumentCommand \@@_rotate: { O { } }
{ 

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).

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).

Indeed, we want that the user may use the command \line in \CodeAfter with LaTeX counters in the arguments — with the command \value.
The test of \texttt{measuring} is a security (cf. question 686649 on TeX StackExchange).

The following command \textit{must} be protected (it's used in the construction of \texttt{\@@_draw_line_ii:nn}).

The commands \texttt{\Ldots}, \texttt{\Cdots}, \texttt{\Vdots}, \texttt{\Ddots}, and \texttt{\Iddots} don't use this command because they have to do other settings (for example, the diagonal lines must be parallelized).

\section{The command \texttt{\RowStyle}}

\texttt{\g_@@_row_style_tl} may contain several instructions of the form: \texttt{\@@_if_row_less_than:nn \{ number \} \{ instructions \}}

Then, \texttt{\g_@@_row_style_tl} will be inserted in all the cells of the array (and also in both components of a \texttt{\diagbox} in a cell of in a mono-row block).

The test \texttt{\@@_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 \texttt{nb-rows} of \texttt{\RowStyle}).

That test will be active even in an expandable context because \texttt{\@@_if_row_less_then:nn} is \textit{not} protected.
#1 is the first row after the scope of the instructions in #2

```latex
\cs_new:Npn \@@_if_row_less_than:nn #1 #2
{ \int_compare:nNnT { \c@iRow } < { #1 } { #2 } }
```

`\@\@_put_in_row_style` will be used several times by `\RowStyle`.

```latex
\cs_set_protected:Npn \@@_put_in_row_style:n #1
{ \tl_gput_right:Nx \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 { #1 } \scan_stop: }
\cs_generate_variant:Nn \@@_put_in_row_style:n { e }
```

```latex
\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-bottom-limit .dim_set:N = \l_tmpb_dim ,
cell-space-bottom-limit .value_required:n = true ,
cell-space-limits .meta:n =
  { cell-space-top-limit = #1 ,
    cell-space-bottom-limit = #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:nnTF { #1 } { * } { \int_set:Nn \l_@@_key_nb_rows_int { 500 } }{ \int_set:Nn \l_@@_key_nb_rows_int { #1 } } ,
rowcolor .tl_set:N = \l_tmpa_tl ,
rowcolor .value_required:n = true ,
unknown .code:n = \@@_error:n { Unknown key for RowStyle }
}
```

```latex
\NewDocumentCommand \@@_RowStyle:n { O { } m }
{ \group_begin:
  \tl_clear:N \l_tmpa_tl
  \tl_clear:N \l_@@_color_tl
  \int_set_eq:NN \l_@@_key_nb_rows_int \c\one_int
  \dim_zero:N \l_tmpa_dim
  \dim_zero:N \l_tmpb_dim
  \keys_set:nn { NiceMatrix / RowStyle } { #1 }
  \tl_if_empty:NF \l_tmpa_tl
```

If the key rowcolor has been used.

```
  \tl_if_empty:NF \l_tmpa_tl
  { \l_@@_if_empty:WF \l_tmpa_tl
```

127
First, the end of the current row (we remind that \RowStyle applies to the end of the current row).

\tl_gput_right:Nx \g_@@_pre_code_before_tl
\{
\}

The command \@@_exp_color_arg:No is fully expandable.

\@@_exp_color_arg:No \@@_rectanglecolor \l_tmpa_tl
\{ \int_use:N \c@iRow - \int_use:N \c@jCol \}
\{ \int_use:N \c@iRow - * \}
\}

Then, the other rows (if there is several rows).
\int_compare:nNnT \l_@@_key_nb_rows_int > \c_one_int
\{ \tl_gput_right:Nx \g_@@_pre_code_before_tl
\{ \@@_exp_color_arg:No \@@_rowcolor \l_tmpa_tl
\{ \int_eval:n { \c@iRow + 1 }
- \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int - 1 }
\}
\}
\}
\@@_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
\{ \exp_args:Nx \@@_put_in_row_style:n
\{ \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
\{ \exp_args:Nx \@@_put_in_row_style:n
\{ \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.

5422 \bool_if:NT \l_@@_bold_row_style_bool
5423 {
5424 \@@_put_in_row_style:n
5425 {\exp_not:n
5426 {\if_mode_math:
5427 \c_math_toggle_token
5428 \bfseries \boldmath
5429 \c_math_toggle_token
5430 \else:
5431 \bfseries \boldmath
5432 \fi:
5433 }
5434 }
5435 }
5436 \group_end:
5437 \g_@@_row_style_tl
5438 \ignorespaces

22 Colors of cells, rows and columns

We want to avoid the thin white lines that are shown in some PDF viewers (eg: with the engine MuPDF used by SumatraPDF). That’s why we try to draw rectangles of the same color in the same instruction \pgfusepath{fill} (and they will be in the same instruction fill—coded f—in the resulting PDF).

The commands \@@_rowcolor, \@@_columncolor, \@@_rectanglecolor and \@@_rowlistcolors don’t directly draw the corresponding rectangles. Instead, they store their instructions color by color:

- A sequence \g_@@_colors_seq will be built containing all the colors used by at least one of these instructions. Each color may be prefixed by its color model (eg: [gray]{0.5}).
- For the color whose index in \g_@@_colors_seq is equal to i, a list of instructions which use that color will be constructed in the token list \g_@@_color_i_tl. In that token list, the instructions will be written using \@@_cartesian_color:nn and \@@_rectanglecolor:nn.

#1 is the color and #2 is an instruction using that color. Despite its name, the command \@@_add_to_colors_seq:nn doesn’t only add a color to \g_@@_colors_seq: it also updates the corresponding token list \g_@@_color_i_tl. We add in a global way because the final user may use the instructions such as \cellcolor in a loop of pgffor in the \CodeBefore (and we recall that a loop of pgffor is encapsulated in a group).

\cs_new_protected:Npn \@@_add_to_colors_seq:nn #1 #2
\{Firt, we look for the number of the color and, if it’s found, we store it in \l_tmpa_int. If the color is not present in \g_@@_colors_seq, \l_tmpa_int will remain equal to 0.
\int_zero:N \l_tmpa_int
\}
\str_if_in:nF { #1 } { !! }
\{\seq_map_indexed_inline:Nn \g_@@_colors_seq
\{ \tl_if_eq:nTF { #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:cx \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:cx \g_@@_color_ \int_use:N \l_tmpa_int_ \tl \#2
\cs_generate_variant:Nn \@@_add_to_colors_seq:nn { e n }
\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} }
{ \pgfpoint{\l_@@_tab_rounded_corners_dim}{\l_@@_tab_rounded_corners_dim} }
\group_end:

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
{ \l_@@_qpoint:n {row-1}}
{ \pgfpoint{0.5\arrayrulewidth}{\c_zero_dim}}
}
{ \pgfpointadd
{ \l_@@_qpoint:n
{\int_eval:n{\int_max:nn\c@iRow\c@jCol+1}}}
{ \pgfpoint{\c_zero_dim}{0.5\arrayrulewidth}}
}

} { \pgfpathrectanglecorners
{ \l_@@_qpoint:n {row-1}}
{ \pgfpointadd
{ \l_@@_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 \texttt{\pgfsetcornersarced}.

The macro \texttt{\@@\_actually\_color:} will actually fill all the rectangles, color by color (using the sequence \texttt{\l_{\@@\_color\_seq}} and all the token lists of the form \texttt{\l_{\@@\_color\_i\_tl}}).

```
\cs_new_protected:Npn \@@\_actually\_color:
{\pgfpicture
\pgf@relevantforpicturesizefalse
```

If the final user has used the key \texttt{rounded-corners} for the environment \{\texttt{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 \texttt{opacity} in its optional argument (between square brackets) and (of course) then apply the command \texttt{\color}.

```
\cs\new_protected:Npn \@@\_color\_opacity
{\peek\_meaning:NTF [ 
{\@@\_color\_opacity:w}
{\@@\_color\_opacity:w [ ]}
}
```

The command \texttt{\@@\_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 currying.

```
\cs_new_protected:Npn \@@\_color\_opacity:w [ #1 ]
{\tl\_clear:N \l_{\@@\_color\_opacity\_w\_l\_tl}
\keys\_set\_known:nNn { nicematrix / color\_opacity } { #1 } \l_{\@@\_color\_opacity\_w\_l\_tl}
\l_{\@@\_color\_opacity\_w\_l\_tl} (if not empty) is now the opacity and \l_{\@@\_color\_opacity\_w\_l\_tl} (if not empty) is now the colorimetric space.
```

```
\tl\_if\_empty:NF \l_{\@@\_color\_opacity\_w\_l\_tl}\ { \exp\_args:No \pgfsetfillopacity \l_{\@@\_color\_opacity\_w\_l\_tl}
\tl\_if\_empty:NF \l_{\@@\_color\_opacity\_w\_l\_tl}
{\@declaredcolor}
{\use:e \ { \exp\_not:N \@undeclaredcolor [ \l_{\@@\_color\_opacity\_w\_l\_tl} ]}}
```

The following set of keys is used by the command \texttt{\@@\_color\_opacity:wn}.

```
\keys\_define:nn { nicematrix / color\_opacity }
{ }
```
\cs_new_protected:Npn \@@_cartesian_color:nn #1 #2
\{\cs_set_nopar:Npn \l_@@_rows_tl { #1 } \cs_set_nopar:Npn \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.
The command `\@@_cartesian_path:N` takes in two implicit arguments: `\l_@@_cols_tl` and `\l_@@_rows_tl`.

Here is an example:

```
\@@_cellcolor[rgb]{0.5,0.5,0}{{2-3,3-4,4-5,5-6}}
```

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 } }
```

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 `\@@_rowlistcolors` appears as a special case of `\@@_rowlistcolors`.
The group is for the options. \l@@_colors_seq will be the list of colors.

\group_begin:
\seq_clear_new:N \l@@_colors_seq
\seq_set_split:Nnn \l@@_colors_seq { , } { #3 }
\tl_clear_new:N \l@@_cols_tl
\cs_set_nopar:Npn \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 completely in the “first column” (number 0) or in the “last column” and that’s why we filter the sequence of the blocks (in a the sequence \l_tmpa_seq).
\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 }
\pgfpicture
\clist_map_inline:nn { #2 } {
\cs_set_nopar:Npn \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@@_tmpc_int will be the index of the loop over the rows.
\int_set:Nn \l@@_tmpc_int \l_tmpb_tl
\int_do_until:nNnn \l@@_tmpc_int > \l@@_tmpc_int { \tl_set:No \l@@_color_tl { \int_use:N \l@@_tmpc_int - \int_use:N \l@@_tmpc_int } \tl_clear_new:N \l@@_color_tl
\tl_set:Nx \l@@_color_tl { #3 }
\tl_set:No \l@@_rows_tl { \int_use:N \l@@_tmpc_int - \int_use:N \l@@_tmpc_int }
\l@@_tmpc_tl will be the color that we will use.
\tl_clear_new:N \l@@_color_tl
\tl_set:Nx \l@@_color_tl { #4 }
}\l@@_tmpc_tl will be the color that we will use.
\@@_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
\group_end:

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
{
\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 an 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:nNnT { #2 } > \c@jCol
\prg_return_false:
\prg_return_true:
\prg_return_false:
}

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:nNnT { #1 } > \l_tmpa_int
\prg_return_false:
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:
\{ \seq_if_empty:NTF \l_@@_corners_cells_seq
\{ \@@_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 } }
\}
\clist_map_inline:Nn \l_@@_cols_tl
\{ \cs_set_nopar:Npn \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
\{ \cs_set_nopar:Npn \l_tmpa_tl { 1 } \}
\{ \tl_if_eq:NNT \l_tmpa_tl \c_@@_star_tl
\{ \cs_set_nopar:Npn \l_tmpa_tl { 1 } \}
\}
\int_compare:nNnT \l_tmpa_tl > \g_@@_col_total_int
\{ \int_compare:nNn \l_tmpa_tl > \g_@@_col_total_int \}
\l_@@_tmpc_tl will contain the number of column.

\l_@@_tmppc_tl will contain the number of column.
We begin the loop over the rows.

\clist_map_inline:Nn \l_@@_rows_tl
{
    \cs_set_nopar:Npn \l_tmpa_tl { \textbf{\int_eval:n { \l_@@_corners_cells_seq } } }
    \tl_if_in:NnTF \l_tmpa_tl { - }
    { \@@_cut_on_hyphen:w \textbf{\int_eval:n { \l_@@_corners_cells_seq } } \q_stop }
    { \@@_cut_on_hyphen:w \textbf{\int_eval:n { \l_@@_corners_cells_seq } } - \textbf{\int_eval:n { \l_@@_corners_cells_seq } } \q_stop }
    \tl_if_empty:NTF \l_tmpa_tl
    { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
    { \tl_if_eq:NNT \l_tmpa_tl \c_@@_star_tl
    { \cs_set_nopar:Npn \l_tmpa_tl { 1 } }
    }
    \tl_if_empty:NTF \l_tmpb_tl
    { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }
    { \tl_if_eq:NNT \l_tmpb_tl \c_@@_star_tl
    { \tl_set:No \l_tmpb_tl { \int_use:N \c@iRow } }
    }
    \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 { @@_\l_@@_tmpa_tl _ \l_@@_corners_seq _ nocolor }
{
    \@@_qpoint:n { \textbf{\int_eval:n { \l_@@_corners_seq } } \l_@@_tmpc_tl _ row - \textbf{\int_eval:n { \l_@@_corners_seq } } \l_@@_tmpd_tl _ nocolor }
    \dim_set:Nn \l_@@_tmpa_dim { \pgf@y + 0.5 \arrayrulewidth }
    \@@_qpoint:n { \textbf{\int_eval:n { \l_@@_corners_seq } } \l_@@_tmpb_tl _ col - \textbf{\int_eval:n { \l_@@_corners_seq } } \l_@@_tmpb_tl _ nocolor }
    \dim_set:Nn \l_@@_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
}
\cs_if_exist:cF { @@_\l_@@_tmpa_tl _ \l_@@_corners_seq _ nocolor }
{
    \@@_cartesian_path_normal_ii:
    \@@@@_\l_@@_corners_seq _ \l_@@_corners_seq _ nocolor }
\cs_new_protected:Npn \@@_cartesian_path_normal_ii:
{
    \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
    \@@_expand_clist:NN \l_@@_rows_tl \c@iRow
    \@@_expand_clist:NN \l_@@_corners_cells_seq \c@iRow \c@jCol
    \cs_if_exist:cF { @@_\l_@@_tmpa_tl _ \l_@@_corners_seq _ nocolor }
    \{ \@@_qpoint:n { \textbf{\int_eval:n { \l_@@_corners_seq } } \l_@@_tmpc_tl _ row - \textbf{\int_eval:n { \l_@@_corners_seq } } \l_@@_tmpd_tl _ nocolor }
    \dim_set:Nn \l_@@_tmpa_dim { \pgf@y + 0.5 \arrayrulewidth }
    \@@_qpoint:n { \textbf{\int_eval:n { \l_@@_corners_seq } } \l_@@_tmpb_tl _ col - \textbf{\int_eval:n { \l_@@_corners_seq } } \l_@@_tmpb_tl _ nocolor }
    \dim_set:Nn \l_@@_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
}
\cs_if_exist:cF { @@_\l_@@_tmpa_tl _ \l_@@_corners_seq _ nocolor }
{
    \@@_cartesian_path_normal_ii:
    \@@@@_\l_@@_corners_seq _ \l_@@_corners_seq _ nocolor }
\cs_new_protected:Npn \@@_cartesian_path_normal_ii:
{
    \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
    \@@_expand_clist:NN \l_@@_rows_tl \c@iRow
    \@@_expand_clist:NN \l_@@_corners_cells_seq \c@iRow \c@jCol
    \cs_if_exist:cF { @@_\l_@@_tmpa_tl _ \l_@@_corners_seq _ nocolor }
    \{ \@@_qpoint:n { \textbf{\int_eval:n { \l_@@_corners_seq } } \l_@@_tmpc_tl _ row - \textbf{\int_eval:n { \l_@@_corners_seq } } \l_@@_tmpd_tl _ nocolor }
    \dim_set:Nn \l_@@_tmpa_dim { \pgf@y + 0.5 \arrayrulewidth }
    \@@_qpoint:n { \textbf{\int_eval:n { \l_@@_corners_seq } } \l_@@_tmpb_tl _ col - \textbf{\int_eval:n { \l_@@_corners_seq } } \l_@@_tmpb_tl _ nocolor }
    \dim_set:Nn \l_@@_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
}
\cs_if_exist:cF { @@_\l_@@_tmpa_tl _ \l_@@_corners_seq _ nocolor }
{
    \@@_cartesian_path_normal_ii:
    \@@@@_\l_@@_corners_seq _ \l_@@_corners_seq _ nocolor }
\cs_new_protected:Npn \@@_cartesian_path_normal_ii:
{
    \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
    \@@_expand_clist:NN \l_@@_rows_tl \c@iRow
    \@@_expand_clist:NN \l_@@_corners_cells_seq \c@iRow \c@jCol
    \cs_if_exist:cF { @@_\l_@@_tmpa_tl _ \l_@@_corners_seq _ nocolor }
    \{ \@@_qpoint:n { \textbf{\int_eval:n { \l_@@_corners_seq } } \l_@@_tmpc_tl _ row - \textbf{\int_eval:n { \l_@@_corners_seq } } \l_@@_tmpd_tl _ nocolor }
    \dim_set:Nn \l_@@_tmpa_dim { \pgf@y + 0.5 \arrayrulewidth }
    \@@_qpoint:n { \textbf{\int_eval:n { \l_@@_corners_seq } } \l_@@_tmpb_tl _ col - \textbf{\int_eval:n { \l_@@_corners_seq } } \l_@@_tmpb_tl _ nocolor }
    \dim_set:Nn \l_@@_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
}
\cs_if_exist:cF { @@_\l_@@_tmpa_tl _ \l_@@_corners_seq _ nocolor }
{
    \@@_cartesian_path_normal_ii:
    \@@@@_\l_@@_corners_seq _ \l_@@_corners_seq _ nocolor }
\cs_new_protected:Npn \@@_cartesian_path_normal_ii:
{
    \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
    \@@_expand_clist:NN \l_@@_rows_tl \c@iRow
    \@@_expand_clist:NN \l_@@_corners_cells_seq \c@iRow \c@jCol
    \cs_if_exist:cF { @@_\l_@@_tmpa_tl _ \l_@@_corners_seq _ nocolor }
    \{ \@@_qpoint:n { \textbf{\int_eval:n { \l_@@_corners_seq } } \l_@@_tmpc_tl _ row - \textbf{\int_eval:n { \l_@@_corners_seq } } \l_@@_tmpd_tl _ nocolor }
    \dim_set:Nn \l_@@_tmpa_dim { \pgf@y + 0.5 \arrayrulewidth }
    \@@_qpoint:n { \textbf{\int_eval:n { \l_@@_corners_seq } } \l_@@_tmpb_tl _ col - \textbf{\int_eval:n { \l_@@_corners_seq } } \l_@@_tmpb_tl _ nocolor }
    \dim_set:Nn \l_@@_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
}
\cs_if_exist:cF { @@_\l_@@_tmpa_tl _ \l_@@_corners_seq _ nocolor }
{
    \@@_cartesian_path_normal_ii:
    \@@@@_\l_@@_corners_seq _ \l_@@_corners_seq _ nocolor }

\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 { @@ _ ####1 _ ##1 _ nocolor }
\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:cpn { @@ _ ##1 _ ####1 _ nocolor } { } }

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
\bool_set_true:N \l_@@_nocolor_used_bool
\@@_expand_clist:NN \l_@@_cols_tl \c@jCol
\@@_expand_clist:NN \l_@@_rows_tl \c@iRow

138
When the user uses the key `color-inside`, the following command will be linked to \texttt{\cellcolor} in the tabular.

\begin{verbatim}
\NewDocumentCommand \@@_cellcolor_tabular { O { } m }
  { \@@_test_color_inside:
    \tl_gput_right:Nx \g_@@_pre_code_before_tl
    \@@_rectanglecolor \[ #1 \] { \exp_not:n { #2 } }
    { \int_use:N \c@iRow - \int_use:N \c@jCol }

\end{verbatim}

We must not expand the color (#2) because the color may contain the token `\texttt{!}` which may be activated by some packages (ex.: \texttt{babel} with the option \texttt{french} on latex and pdflatex).

\begin{verbatim}
\@@_cellcolor \[ #1 \] { \exp_not:n { #2 } }
\end{verbatim}

\begin{verbatim}
\@@_rectanglecolor \[ #1 \] { \exp_not:n { #2 } }
\end{verbatim}

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

\begin{verbatim}
\NewDocumentCommand \@@_rowcolor_tabular { O { } m }
  { \@@_test_color_inside:
    \tl_gput_right:Nx \g_@@_pre_code_before_tl
    \@@_rectanglecolor \[ #1 \] { \exp_not:n { #2 } }
    { \int_use:N \c@iRow - \int_use:N \c@jCol }

\end{verbatim}

The last argument (an optional argument between square brackets is taken by curryfication).

\begin{verbatim}
\NewDocumentCommand \@@_rowlistcolors_tabular { O { } m O { } }
  { \@@_test_color_inside:
    \peek_remove_spaces:n { \@@_rowlistcolors_tabular:nnn { #1 } { #2 } { #3 } }

\end{verbatim}

\begin{verbatim}
\cs_new_protected:Npn \@@_rowlistcolors_tabular:nnn #1 #2 #3
  { A use of \texttt{\rowlistcolors} in the tabular erases the instructions \texttt{\rowlistcolors} which are in force. However, it’s possible to put several instructions \texttt{\rowlistcolors} in the same row of a tabular: it may be useful when those instructions \texttt{\rowlistcolors} concerns different columns of the tabular (thanks to the key \texttt{cols} of \texttt{\rowlistcolors}). That’s why we store the different instructions \texttt{\rowlistcolors} which are in force in a sequence \texttt{\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 \texttt{\g_tmpa_seq}.

\seq_gclear:N \g_tmpa_seq
\seq_map_inline:Nn \g_@@_rowlistcolors_seq
  { \@@_rowlistcolors_tabular_i:nnnn ##1 }
\seq_gset_eq:NN \g_@@_rowlistcolors_seq \g_tmpa_seq

\end{verbatim}

The braces around #2 and #3 are mandatory.
Now, we add to the sequence \texttt{/g@@rowlistcolors_seq} (which is the list of the commands \texttt{/rowlistcolors} which are in force) the current instruction \texttt{/rowlistcolors}.

\begin{verbatim}
\seq_gput_right:Nx \g@@rowlistcolors_seq
  { \int_use:N \c@iRow }
  { \exp_not:n \{ \textit{#1} \} }
  { \exp_not:n \{ \textit{#2} \} }
  { \textit{restart}, \textit{cols} = \int_use:N \c@jCol - , \exp_not:n \{ \textit{#3} \} }
\}
\end{verbatim}

The following command will be applied to each component of \texttt{/g@@rowlistcolors_seq}. Each component of that sequence is a kind of 4-uple of the form \{\textit{#1}\textit{#2}\textit{#3}\textit{#4}\}.

- \textit{#1} is the number of the row where the command \texttt{/rowlistcolors} has been issued.
- \textit{#2} is the colorimetric space (optional argument of the \texttt{/rowlistcolors}).
- \textit{#3} is the list of colors (mandatory argument of \texttt{/rowlistcolors}).
- \textit{#4} is the list of \texttt{key=value} pairs (last optional argument of \texttt{/rowlistcolors}).

\begin{verbatim}
\cs_new_protected:Npn \@@rowlistcolors_tabular_i:nnnn #1 #2 #3 #4
  { \int_compare:nNnTF \{ \textit{#1} \} = \c@iRow
    { \seq_gput_right:Nn \g_tmpa_seq { \{ \textit{#1} \} \{ \textit{#2} \} \{ \textit{#3} \} \{ \textit{#4} \} } }

    \tl_gput_right:Nx \g_@@pre_code_before_tl
    { \@@rowlistcolors \[ \exp_not:n \{ \textit{#2} \} \]
      \{ \textit{#1} - \int_eval:n \{ \c@iRow - \textit{1} \} \}
      \{ \exp_not:n \{ \textit{#3} \} \}
      \{ \exp_not:n \{ \textit{#4} \} \}
  }
\}
\end{verbatim}

We (temporary) keep in memory in \texttt{/g@tmp@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).

\begin{verbatim}
\seq_gput_right:Nn \g@tmp@seq \{ \{ \textit{#1} \} \{ \textit{#2} \} \{ \textit{#3} \} \{ \textit{#4} \} \}
\end{verbatim}

The following command will be used at the end of the tabular, just before the execution of the \texttt{/g@@pre_code_before_tl}. It clears the sequence \texttt{/g@@rowlistcolors_seq} of all the commands \texttt{/rowlistcolors} which are (still) in force.

\begin{verbatim}
\cs_new_protected:Npn \@@clear_rowlistcolors_seq:
  { \seq_map_inline:Nn \g@@rowlistcolors_seq
    \{ \@@rowlistcolors_tabular_ii:nnnn \#1 \#2 \#3 \#4 \}

    \seq_gclear:N \g@@rowlistcolors_seq
  }
\end{verbatim}

The first mandatory argument of the command \texttt{\@@rowlistcolors} which is written in the \texttt{pre/\CodeBefore} is of the form \textit{i}: it means that the command must be applied to all the rows from the row \textit{i} until the end of the tabular.

\begin{verbatim}
\NewDocumentCommand \@@columncolor_preamble \{ O { } \} m
  { }
\end{verbatim}
23 The vertical and horizontal rules

OnlyMainNiceMatrix

We give to the user the possibility to define new types of columns (with \newcolumntype of array) for special vertical rules (e.g. rules thicker than the standard ones) which will not extend in the potential exterior rows of the array.

We provide the command \OnlyMainNiceMatrix in that goal. However, that command must be no-op outside the environments of nicematrix (and so the user will be allowed to use the same new type of column in the environments of nicematrix and in the standard environments of array). That's why we provide first a global definition of \OnlyMainNiceMatrix.

\cs_set_eq:NN \OnlyMainNiceMatrix \use:n

Another definition of \OnlyMainNiceMatrix will be linked to the command in the environments of nicematrix. Here is that definition, called \@\OnlyMainNiceMatrix:n.

\cs_new_protected:Npn \@\OnlyMainNiceMatrix: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.

Remember that $\c@iRow$ is not always inferior to $\l_@@_last_row_int$ because $\l_@@_last_row_int$ may be equal to $-2$ or $-1$ (we can't write $\int_compare:nNnT \c@iRow < \l_@@_last_row_int$).

General system for drawing rules

When a command, environment or “subsystem” of nicematrix wants to draw a rule, it will write in the internal $\CodeAfter$ a command $\@@_{vline:n}$ or $\@@_{hline:n}$. Both commands take in as argument a list of key=value pairs. That list will first be analyzed with the following set of keys. However, unknown keys will be analyzed further with another set of keys.

It’s possible that the rule won’t be drawn continuously from start ot end because of the blocks (created with the command $\Block$), the virtual blocks (created by $\Cdots$, etc.), etc. That’s why an analyse is done and the rule is cut in small rules which will actually be drawn. The small continuous rules will be drawn by $\@@_{vline_ii}$ and $\@@_{hline_ii}$: Those commands use the following set of keys.
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]`.

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.

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
\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
\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 |B|lock or a virtual block corresponding to a dotted line, created by |D|ots, |V|dots, 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
\seq_map_inline:Nn \g_\pos_of_xdots_seq
\seq_map_inline:Nn \g_\pos_of_stroken_blocks_seq
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.

\begin{verbatim}
\int_compare:nNnT \l_@@_local_start_int > \c_zero_int
\{ \l_set:Nn \l_@@_local_start_int \l_tmpa_tl \}
\}
\int_compare:nNnTF \l_tmpb_tl = { \int_eval:n { \c@jCol + 1 } }
\{ \int_compare:nNnT \l_@@_local_start_int > \c_zero_int
\int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
\@@_vline_ii: \}
\}
\end{verbatim}

\begin{verbatim}
\cs_new_protected:Npn \@@_vline_ii:
\{ \tl_clear:N \l_@@_tikz_rule_tl \keys_set:nV { 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:
\}
\}
\end{verbatim}

\begin{verbatim}
\cs_new_protected:Npn \@@_vline_iii:
\{ \tl_clear:N \l_@@_tikz_rule_tl \keys_set:nV { 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:
\}
\}
\end{verbatim}

\begin{verbatim}
\cs_new_protected:Npn \@@_vline_iv:
\{ \tl_clear:N \l_@@_tikz_rule_tl \keys_set:nV { NiceMatrix / RulesBis } \l_@@_other_keys_tl
\bool_if:NTF \l_@@_dotted_bool \@@_vline_v:
\{ \tl_if_empty:NTF \l_@@_tikz_rule_tl \@@_vline_iii:
\@@_vline_v:
\}
\}
\end{verbatim}

\begin{verbatim}
\cs_new_protected:Npn \@@_vline_v:
\{ \tl_clear:N \l_@@_tikz_rule_tl \keys_set:nV { NiceMatrix / RulesBis } \l_@@_other_keys_tl
\bool_if:NTF \l_@@_dotted_bool \@@_vline_v:
\{ \tl_if_empty:NTF \l_@@_tikz_rule_tl \@@_vline_iii:
\@@_vline_v:
\}
\}
\end{verbatim}

144
First the case of a standard rule: the user has not used the key `dotted` nor the key `tikz`.

```latex
\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_@@_tmpc_dim { \pgf@y
\bool_lazy_all:nT
{ \int_compare_p:nNn \l_@@_multiplicity_int > \c_one_int
{ ! \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 )
}
\pgfsetrectanglecorners
\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_@@_tmpd_dim \l_@@_tmpc_dim }
\pgfsetlinecap { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
\pgfsetlinewidth { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
\pgfusepathqstroke
\endpgfpicture
}
```

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

```latex
\cs_new_protected:Npn \@@_vline_iv:
{\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_@@_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 )
}
\pgfsetrectanglecorners
\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_@@_tmpd_dim \l_@@_tmpc_dim }
\pgfsetlinecap { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
\pgfsetlinewidth { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
\pgfusepathqstroke
\endpgfpicture
}
The following code is for the case when the user uses the key `tikz`.

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).

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).

```latex
\cs_new_protected:Npn \__\_draw_vlines: 
\begin {tikzpicture} 
\begin {scope} 
```
```
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_zero_new:N \l_@@_end_int
\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:
\{
\int_zero_new:N \l_@@_local_start_int
\int_zero_new:N \l_@@_local_end_int
\l_tmpa_tl
is the number of row and \l_tmpb_tl the number of column. When we have found a column corresponding to a rule to draw, we note its number in \l_@@_local_start_int.

\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 \Dots, \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
\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
{ \int_set:Nn \l_@@_local_end_int \l_@@_end_int
\@@_hline_ii:
\int_zero:N \l_@@_local_start_int
} } { }
\int_compare:nNnT \l_@@_local_start_int > \c_zero_int
{ \int_compare:nNnT \l_@@_local_start_int > \c_zero_int
{ \int_compare:nNnT \l_@@_local_start_int > \c_zero_int
{ \int_set:Nn \l_@@_local_end_int { \l_tmpb_tl - 1 }
\@@_hline_i:
} } } }
\}

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_i:
\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_i:
} }
First the case of a standard rule (without the keys `dotted` and `tikz`).
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}

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}
For reasons purely aesthetic, we do an adjustment in the case of a rounded bracket. The correction by 0.5 $\ell_\text{xdots_inter_dim}$ is ad hoc for a better result.

The following code is for the case when the user uses the key \texttt{tikz} (in the definition of a customized rule by using the key \texttt{custom-line}).

By default, the color defined by \texttt{arrayrulecolor} or by \texttt{rules/color} will be used, but it’s still possible to change the color by using the key \texttt{color} or, of course, the key \texttt{color} inside the key \texttt{tikz} (that is to say the key \texttt{color} provided by \texttt{PGF}).

The command \texttt{\@\_draw_hlines}: draws all the horizontal rules excepted in the blocks (even the virtual blocks determined by commands such as \texttt{\_dots} and in the corners — if the key \texttt{corners} is used).
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:Npn \@@_Hline_iv:nn #1 #2
{ \@@_compute_rule_width:n { multiplicity = #1 , #2 }
\skip_vertical:N \l_@@_rule_width_dim
\tl_gput_right:Nx \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
\bool_lazy_all:nTF
{ \@@_custom_line:nn { \l_@@_command_str , \l_@@_ccommand_str , \l_@@_letter_str , \l_@@_other_keys_tl }
\keys_set_known:nN \l_@@_other_keys_tl
\NiceMatrix / custom-line } { #1 } \l_@@_other_keys_tl
\}

If the final user only wants to draw horizontal rules, he does not need to specify a letter (for the vertical rules in the preamble of the array). On the other hand, if he only wants to draw vertical rules, he does not need to define a command (which is the tool to draw horizontal rules in the array). Of course, a definition of custom lines with no letter and no command would be point-less.
The following flags will be raised when the keys `tikz`, `dotted` and `color` are used (in the `custom-line`).

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.
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 \customline) 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_required:n = true ,
  total-width .code:n = \dim_set:Nn \l_@@_rule_width_dim { #1 } ,
  total-width .value_required:n = true ,
  width .code:n = \dim_set:Nn \l_@@_rule_width_dim { #1 } ,
  width .value_required:n = true ,
  sep-color .code:n = \dim_set:Nn \l_@@_rule_width_dim { #1 } ,
  sep-color .value_required:n = true ,
  unknown .code:n = \@@_error:n { Unknown key for custom-line } 
\}

The following keys will indicate whether the keys dotted, tikz and color are used in the use of a \customline.

\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 ,
  tikz .value_required:n = true ,
  total-width .code:n = \dim_set:Nn \l_@@_total_width_bool { #1 } ,
  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: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).
Here, we need an expandable command since it begins with an `\noalign`.

\exp_args:Nc \NewExpandableDocumentCommand
\{ nicematrix - \l_@@_ccommand_str \}
\{ 0 \{ } m \}
\{
\noalign
\{ \@@_compute_rule_width:n { #1 , ##1 } \skip_vertical:n { \l_@@_rule_width_dim } \clist_map_inline:nn
\{ \#2 \}
\{ \l_@@_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 \l_@@_c_custom_line_i:nn #1 #2
\{
\str_if_in:nnTF { #2 } { - } { \@@_cut_on_hyphen:w #2 \q_stop } { \@@_cut_on_hyphen:w #2 - #2 \q_stop }
\tl_gput_right:Nx \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 \l_@@_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 }
\}{ \l_@@_total_width_bool
\bool_if:NTF \l_@@_tikz_rule_bool
\dim_set:Nn \l_@@_rule_width_dim
\{ \arrayrulewidth * \l_@@_multiplicity_int
+ \doublerulesep * ( \l_@@_multiplicity_int - 1 ) \}
\}
\}
\}
\cs_new_protected:Npn \l_@@_v_custom_line:n #1
\{
\l_@@_compute_rule_width:n { #1 }
\}

154
In the following line, the \texttt{\dim_use:N} is mandatory since we do an expansion.

\begin{verbatim}
\tl_gput_right:Nx \g_@@_array_preamble_tl
\{ { \exp_not:N ! \{ \skip_horizontal:n \{ \dim_use:N \l_@@_rule_width_dim \} \} }
\tl_gput_right:Nx \g_@@_pre_code_after_tl
\{ \@@_vline:n
\{ #1 ,
   position = \int_eval:n \{ \c@jCol + 1 \} ,
   total-width = \dim_use:N \l_@@_rule_width_dim
\} \@@_rec_preamble:n
\} \@@_custom_line:n
\{ letter = : , command = hdottedline , ccommand = cdottedline , dotted \}
\end{verbatim}

The key \texttt{hlvlines}

The following command tests whether the current position in the array (given by \texttt{\l_tmpa_tl} for the row and \texttt{\l_tmpb_tl} for the column) would provide an horizontal rule towards the right in the block delimited by the four arguments \texttt{#1}, \texttt{#2}, \texttt{#3} and \texttt{#4}. If this rule would be in the block (it must not be drawn), the boolean \texttt{\l_tmpa_bool} is set to false.

\begin{verbatim}
\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 \}
\}
\}
\end{verbatim}

The same for vertical rules.

\begin{verbatim}
\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 \}
\}
\}
\end{verbatim}

\begin{verbatim}
\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:nNnT \l_tmpa_tl = { #1 }
\{ \bool_gset_false:N \g_tmpa_bool \}
\}
\end{verbatim}
24 The empty corners

When the key `corners` is raised, the rules are not drawn in the corners; they are not colored and `\TikzEveryCell` does not apply. Of course, we have to compute the corners before we begin to draw the rules.

```latex
\cs_new_protected:Npn \@@_compute_corners: 
{ ...
```

The sequence `\l_@@_corners_cells_seq` will be the sequence of all the empty cells (and not in a block) considered in the corners of the array.

```latex
\seq_clear_new:N \l_@@_corners_cells_seq
\clist_map_inline:Nn \l_@@_corners_clist
{ ...
```

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

```latex
\seq_if_empty:NF \l_@@_corners_cells_seq
{ ...
```

You write on the `aux` file the list of the cells which are in the (empty) corners because you need that information in the `\CodeBefore` since the commands which color the `rows`, `columns` and `cells` must not color the cells in the corners.

```latex
\tl_gput_right:Nx \g_@@_aux_tl
{ ...
```

```latex
\seq_set_from_clist:Nn \exp_not:N \l_@@_corners_cells_seq
```

156
“Computing a corner” is determining all the empty cells (which are not in a block) that belong to that corner. These cells will be added to the sequence \l_@@_corners_cells_seq.

The six arguments of \@@_compute_a_corner:nnnnnn are as follow:

- \#1 and \#2 are the number of row and column of the cell which is actually in the corner;
- \#3 and \#4 are the steps in rows and the step in columns when moving from the corner;
- \#5 is the number of the final row when scanning the rows from the corner;
- \#6 is the number of the final column when scanning the columns from the corner.

\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.

1. 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 }
{
\@@_test_if_cell_in_a_block:nn { ##1 } { \int_eval:n { \#2 } }
\bool_lazy_or:nnTF
\cs_if_exist_p:c
{ pgf @ sh @ ns @ \@@_env: - ##1 - \int_eval:n { \#2 } }
\l_tmpb_bool
{ \bool_set_true:N \l_tmpa_bool }
\bool_if:NF \l_tmpa_bool
{ \int_set:Nn \l_@@@last_empty_row_int { ##1 } }
}

2. 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 }
{
\@@_test_if_cell_in_a_block:nn { \int_eval:n { \#1 } } { ##1 }
\bool_lazy_or:nnTF
\cs_if_exist_p:c
{ pgf @ sh @ ns @ \@@_env: - \int_eval:n { \#1 } - ##1 }
\l_tmpb_bool
{ \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

\@@_test_if_cell_in_a_block:nn { ##1 } { ####1 }
\bool_lazy_or:nnTF \l_tmpb_bool
{ \cs_if_exist_p:c { pgf @ sh @ ns \@@_env: - ##1 - ####1 } }
{ \bool_set_true:N \l_tmpa_bool }

\bool_if:NF \l_tmpa_bool
{ \int_set:Nn \l_@@_last_empty_column_int { ####1 }
 \seq_put_right:Nn \l_@@_corners_cells_seq { ##1 - ####1 } }

The following macro tests whether a cell is in (at least) one of the blocks of the array (or in a cell with a \diagbox).

The flag \l_tmpb_bool will be raised if the cell #1–#2 is in a block (or in a cell with a \diagbox).

\cs_new_protected:Npn \@@_test_if_cell_in_block:nnnnnnnn #1 #2 #3 #4 #5 #6 #7
{ \int_set:Nn \l_tmpa_int { #1 }
 \int_set:Nn \l_tmpb_int { #2 }
 \bool_set_false:N \l_tmpb_bool
 \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
 { \@@_test_if_cell_in_block:nnnnnnn #1 #2 #3 #4 #5 #6 #7 #1 }
 \cs_set_protected:Npn \@@_test_if_cell_in_block:nnnnnnnn #1 #2 #3 #4 #5 #6 #7 #1
{ \int_compare:nNnF \l_@@_corners_cells_seq { #1 } \l_@@_corners_cells_seq
 { \int_compare:nNnF \l_@@_corners_cells_seq { #2 } \l_@@_corners_cells_seq
   { \int_compare:nNnF \l_@@_corners_cells_seq { #3 } \l_@@_corners_cells_seq
     { \int_compare:nNnF \l_@@_corners_cells_seq { #4 } \l_@@_corners_cells_seq
       { \int_compare:nNnF \l_@@_corners_cells_seq { #5 } \l_@@_corners_cells_seq
         { \int_compare:nNnF \l_@@_corners_cells_seq { #6 } \l_@@_corners_cells_seq
           \bool_set_true:N \l_tmpb_bool
         } } } } } }

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

\begin{verbatim}
\bool_new:N \l_@@_block_auto_columns_width_bool
\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} {
 \exp_args:NNe \dim_set:Nn \l_@@_columns_width_dim
 { \use:c{ @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int} }
 }
}
}
\end{verbatim}

At the end of the environment \texttt{\{NiceMatrixBlock\}}, we write in the main \texttt{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 \texttt{\l_@@\_first\_env\_block\_int}).

\begin{verbatim}
\legacy_if:nTF { measuring@ }{
 \int_gdecr:N \g_@@_NiceMatrixBlock_int
 \bool_if:NT \l_@@_block_auto_columns_width_bool{
 \iow_shipout:Nn \@mainaux \ExplSyntaxOn
 \iow_shipout:Nx \@mainaux{
 \cs_gset:cpn{ @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int}
 \dim_eval:n{ \g_@@_max_cell_width_dim + \arrayrulewidth }
}
 \iow_shipout:Nn \@mainaux \ExplSyntaxOff
}
\ignorespacesafterend
\end{verbatim}

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

\begin{verbatim}
\legacy_if:nTF { measuring@ }{
 \int_gdecr:N \g_@@_NiceMatrixBlock_int
 \bool_if:NT \l_@@_block_auto_columns_width_bool{
 \iow_shipout:Nn \@mainaux \ExplSyntaxOn
 \iow_shipout:Nx \@mainaux{
 \cs_gset:cpn{ \@@_max_\_cell_\_width_ \int_use:N \g_@@_NiceMatrixBlock_int}
 \dim_eval:n{ \g_@@_max_cell_width_dim + \arrayrulewidth }
}
 \iow_shipout:Nn \@mainaux \ExplSyntaxOff
}
\ignorespacesafterend
\end{verbatim}
The extra nodes

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

\cs_generate_variant:Nn \dim_min:nn { v n }
\cs_generate_variant:Nn \dim_max:nn { v n }

The following command is called in \@@_use_arraybox_with_notes_c: just before the construction of the blocks (if the creation of medium nodes is required, medium nodes are also created for the blocks and that construction uses the standard medium nodes).

\cs_new_protected:Npn \@@_create_extra_nodes:
\begin{verbatim}
\bool_if:nTF \l_@@_medium_nodes_bool
\begin{verbatim}
\bool_if:NTF \l_@@_large_nodes_bool
\@@_create_medium_and_large_nodes:
\@@_create_medium_nodes:
\end{verbatim}
\end{verbatim}
\{ \bool_if:NT \l_@@_large_nodes_bool \@@_create_large_nodes: \}
\end{verbatim}
\end{verbatim}
\end{verbatim}

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:
\begin{verbatim}
\int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
\begin{verbatim}
\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 }
\end{verbatim}
\end{verbatim}
\end{verbatim}

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

\int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
\begin{verbatim}
\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 }
\end{verbatim}

160
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.

\[
\begin{align*}
&\text{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 \text{ and } \pgf@y. \\
&\text{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 \text{ and } \pgf@y.
\end{align*}
\]

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

\[
\begin{align*}
&\text{Here is the command } \@@\texttt{\_create\_medium\_nodes:}. \text{ When this command is used, the “medium nodes” are created.}
\end{align*}
\]
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”.

\begin{verbatim}
\cs_set_nopar:Npn \l_@@_suffix_tl { -medium }
\@@_create_nodes:
\endpgfpicture
\end{verbatim}

The command `\@@_create_large_nodes:` must be used when we want to create only the “large nodes” and not the medium ones\(^{14}\). However, the computation of the mathematical coordinates of the “large nodes” needs the computation of the mathematical coordinates of the “medium nodes”. Hence, we use first `\@@_computations_for_medium_nodes:` and then the command `\@@_computations_for_large_nodes:`.

\begin{verbatim}
\cs_set_nopar:Npn \l_@@_suffix_tl { - large }
\@@_create_nodes:
\endpgfpicture
\end{verbatim}

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”.

\begin{verbatim}
\cs_set_nopar:Npn \l_@@_suffix_tl { - medium }
\@@_create_nodes:
\@@_computations_for_large_nodes:
\cs_set_nopar:Npn \l_@@_suffix_tl { - large }
\@@_create_nodes:
\endpgfpicture
\end{verbatim}

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 \c@_@@_col_total_int). Idem for the rows.

\begin{verbatim}
\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
\int_step_variable:nNn { \c@iRow - 1 } \@@_i:
{ \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim } { ( \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } + \dim_use:c { l_@@_row _ \int_eval:n { \@@_i: + 1 } _ max _ dim } ) / 2 }
}
\endpgfpicture
\end{verbatim}

\(^{14}\text{If we want to create both, we have to use } \@@_create_medium_and_large_nodes:\)
\dim_set_eq:cc { l_@@_row_ \@i: _\max_ _dim } { l_@@_row_ \@i: _\min_ _dim }
\int_step_variable:nNn \{ \c@jCol - 1 \} \@@_j:
\dim_set:cn { l_@@_column_ \@@_j: _\max_ _dim } { l_@@_column_ \int_eval:n { \@@_j: + 1 } _\min_ _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_@@_left_margin_dim
\dim_add:cn { l_@@_column_ \int_use:N \c@jCol _\max_ _dim }
\l_@@_right_margin_dim
\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

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

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:nNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
\int_step_variable:nNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:

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

\@@_node_for_multicolumn:nn

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.
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
\begin{Verbatim}
\begin{verbatim}
\end{Verbatim}
\end{Verbatim}
\end{Verbatim}
\end{Verbatim}
\end{Verbatim}
\end{Verbatim}
\end{Verbatim}
\end{Verbatim}
\end{Verbatim}
\end{Verbatim}
\end{Verbatim}
\end{Verbatim}
\end{Verbatim}
\end{Verbatim}
\end{Verbatim}
\end{Verbatim}
\end{Verbatim}
\end{Verbatim}
\end{Verbatim}
\end{Verbatim}
\end{Verbatim}
\end{Verbatim}
\end{Verbatim}
\end{Verbatim}
\end{Verbatim}
\end{Verbatim}
\end{Verbatim}
\end{Verbatim}
\end{Verbatim}
\end{Verbatim}
\end{Verbatim}
\end{Verbatim}
\end{Verbatim}
\end{Verbatim}
\end{Verbatim}
\end{Verbatim}
\end{Verbatim}
\end{Verbatim}
\end{Verbatim}
\end{Verbatim}
\end{Verbatim}
\end{Verbatim}
\end{Verbatim}
\end{Verbatim}
\end{Verbatim}
\end{Verbatim}
\end{Verbatim}
\end{Verbatim}
\end{Verbatim}
\end{Verbatim}
\end{Verbatim}
\end{Verbatim}
\end{Verbatim}
\end{Verbatim}
\end{Verbatim}
\end{Verbatim}
\end{Verbatim}
\end{Verbatim}
\end{Verbatim}
\end{Verbatim}
\end{Verbatim}
\end{Verbatim}
\end{Verbatim}
\end{Verbatim}
\end{Verbatim}
\end{Verbatim}
\end{Verbatim}
\end{Verbatim}
\end{Verbatim}
\end{Verbatim}
\end{Verbatim}
\end{Verbatim}
\end{Verbatim}
\end{Verbatim}
\end{Verbatim}
\end{Verbatim}
\end{Verbatim}
\end{Verbatim}
\end{Ver��}
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).
\cs_new:Npn \@@_Block_i #1-#2 \q_stop { \@@_Block_ii:nnnnn { #1 } { #2 } }
With babel with the key czech, the character - (hyphen) is active. That's why we need a special version. Remark that we could not use a preprocessor in the command \@@_Block: to do the job because the command \@@_Block: is defined with the command \NewExpandableDocumentCommand.
\cs_new:NN \@@_Block_i_czech { \@@_Block_i: } \char_set_catcode_active:N - \cs_new:Npn \@@_Block_i_czech #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=value 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_i: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 } }
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:NV \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 \texttt{\Block} that we will analyze now.

\keys_set_known:nn { NiceMatrix / Block / FirstPass } { #3 }
\tl_set:Nx \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_@@_hpos_cell_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 \texttt{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).

For the blocks mono-column, we will compose right now in a box in order to compute its width and take that width into account for the width of the column. However, if the column is a X column, we should not do that since the width is determined by another way. This should be the same for the \texttt{p}, \texttt{m} and \texttt{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.

\bool_set_false:N \l_@@_X_bool
\bool_if:NT \l_@@_amp_in_blocks_bool
{ \tl_if_in:nnT { #5 } { \& } } \{ \bool_set_true:N \l_tmpa_bool \}
\bool_if:NTF \l_tmpa_bool
{ \exp_args:Nee \@@_Block_vii:nnnnn }
{ \exp_args:Nee \@@_Block_vi:nnnnn }
{ \exp_args:Nee \@@_Block_iv:nnnnn }

166
The following macro is for the case of a \Block which is mono-row or mono-column (or both) and don’t use the key p. In that case, the content of the block is composed right now in a box (because we have to take into account the dimensions of that box for the width of the current column or the height and the depth of the current row). However, that box will be put in the array after the construction of the array (by using PGF) with \@@_draw_blocks: and above all \@@_Block_v: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
\begin{Verbatim}
{ \l_tmpa_int \l_tmpb_int } { #3 } { #4 } { #5 } 
\end{Verbatim}

\hbox_gset:cn
{ g_@@_block_box \int_use:N \g_@@_block_box_int _ box }
The following command will be no-op when \texttt{respect-arraystretch} is in force.

\texttt{\@@_reset_arraystretch:}
\texttt{\dim_zero:N} \texttt{\extrarowheight}

\#4 is the optional argument of the command \texttt{\Block}, provided with the syntax \texttt{<...>}.\footnote{\texttt{\#4}}

We adjust \texttt{\l_@@_hpos_block_str} when \texttt{\rotate} has been used (in the cell where the command \texttt{\Block} is used but maybe in \texttt{\#4}, \texttt{\RowStyle}, \texttt{code-for-first-row}, etc.).\footnote{\texttt{\#4}}

\texttt{\@@_adjust_hpos_rotate:}

The boolean \texttt{\g_@@_rotate_bool} will be also considered \texttt{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 \texttt{\{tabular\}}, an \texttt{\{array\}} or a \texttt{\{minipage\}}.

\texttt{\bool_if:NTF \l_@@_tabular_bool}
\texttt{\bool_lazy_all:nTF}
\texttt{\{ \int_compare_p:nNn { \#2 } = \c_one_int \}}

Remind that, when the column has not a fixed width, the dimension \texttt{\l_@@_col_width_dim} has the conventional value of −1 cm.

\texttt{\{ ! \dim_compare_p:nNn \l_@@_col_width_dim < \c_zero_dim \}}
\texttt{\{ ! \g_@@_rotate_bool \}}
\texttt{\}}

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

\texttt{\{ \use:e}
\texttt{\}}
\texttt{\exp_not:N \begin { minipage }%}
\texttt{[ \str_lowercase:o \l_@@_vpos_block_str ]}
\texttt{\l_@@_col_width_dim}
\texttt{\str_case:on \l_@@_hpos_block_str}
\texttt{\{ c \centering r \raggedleft l \raggedright \}}
\texttt{\}}
\texttt{\#5}
\texttt{\end { minipage } \}}

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

\texttt{\{ \use:e}
\texttt{\}}
\texttt{\exp_not:N \begin { tabular }%}
\texttt{[ \str_lowercase:o \l_@@_vpos_block_str ]}
\texttt{\{ \@ { } \l_@@_hpos_block_str \@ { } \}}
\texttt{\}}
\texttt{\#5}
\texttt{\end { tabular } \}}

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

\texttt{\{ \c_math_toggle_token}
\texttt{\use:e}
\texttt{\}}
\texttt{\exp_not:N \begin { array }%}
\texttt{[ \str_lowercase:o \l_@@_vpos_block_str ]}
\texttt{\{ \@ { } \l_@@_hpos_block_str \@ { } \}}
\texttt{\}}
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.

\int_compare:nNnT { #1 } = \c_one_int
{\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 }
}
}

\seq_gput_right:Nx \g_@@_blocks_seq
{\l_tmpa_tl
\l_@@_hpos_block_str

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
Despite its name the following command rotates the box of the block \textit{but also does vertical adjustment of the baseline of the block.}
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 \texttt{p}. In that case, the content of the block is not composed right now in a box. The composition in a box will be done further, just after the construction of the array (cf. \texttt{\@@_draw_blocks} and above all \texttt{\@@_Block_v:nnnnn}).

#1 is \texttt{i} (the number of rows of the block), #2 is \texttt{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.

\begin{verbatim}
\cs_new_protected:Npn \@@_Block_v:nnnnn #1 #2 #3 #4 #5
\begin{verbatim}
\seq_gput_right:Nx \g_@@_blocks_seq
\l_tmpa_tl { \exp_not:n { #3 } }
\bool_if:NTF \l_@@_tabular_bool \l_@@_tabular_bool
\group_begin:
\@@_reset_arraystretch:
\exp_not:n
\dim_zero:N \extrarowheight
#4
\bool_if:NT \c_@@_testphase_table_bool \tag_stop:n { table }
\use:e
\exp_not:N \begin { tabular } [ \l_@@_vpos_block_str ]
\@ { } \l_@@_hpos_block_str @ { }
\end { tabular }
\c_math_toggle_token
\group_end:
\end{verbatim}
\end{verbatim}

When we are not in an environment \{NiceTabular\} (or similar).

\begin{verbatim}
\@@_reset_arraystretch:
\exp_not:n
\dim_zero:N \extrarowheight
#4
\c_math_toggle_token
\use:e
\exp_not:N \begin { array } [ \l_@@_vpos_block_str ]
\@ { } \l_@@_hpos_block_str @ { }
\end { array }
\c_math_toggle_token
\group_end:
\end{verbatim}

The following command will be no-op when \texttt{respect-arraystretch} is in force.

\begin{verbatim}
\@@_reset_arraystretch:
\exp_not:n
\dim_zero:N \extrarowheight
#4
\bool_if:NT \c_@@_testphase_table_bool \tag_stop:n { table }
\use:e
\exp_not:N \begin { tabular } [ \l_@@_vpos_block_str ]
\@ { } \l_@@_hpos_block_str @ { }
\end { tabular }
\c_math_toggle_token
\group_end:
\end{verbatim}

If the box is rotated (the key \texttt{rotate} may be in the previous #4), the tabular used for the content of the cell will be constructed with a format \texttt{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).

\begin{verbatim}
\bool_if:NT \c_@@_testphase_table_bool \tag_stop:n { table }
\use:e
\exp_not:N \begin { tabular } [ \l_@@_vpos_block_str ]
\@ { } \l_@@_hpos_block_str @ { }
\end { tabular }
\c_math_toggle_token
\group_end:
\end{verbatim}

When we are not in an environment \{NiceTabular\} (or similar).

\begin{verbatim}
\@@_reset_arraystretch:
\exp_not:n
\dim_zero:N \extrarowheight
#4
\c_math_toggle_token
\use:e
\exp_not:N \begin { array } [ \l_@@_vpos_block_str ]
\@ { } \l_@@_hpos_block_str @ { }
\end { array }
\c_math_toggle_token
\group_end:
\end{verbatim}

\end{verbatim}
The following macro is for the case of a `\Block` which uses the key `p`.

```latex
\cs_new_protected:Npn \@@_Block_vi:nnnnn #1 #2 #3 #4 #5
{\seq_gput_right:Nx \g_@@_blocks_seq
 \l_tmpa_tl { \exp_not:n { #3 } }
 { \group_begin:
   \exp_not:n { #4 #5 }
   \group_end:
 }\}
\}
```

The following macro is for the case of a `\Block` which uses the key `p`.

```latex
\cs_new_protected:Npn \@@_Block_vii:nnnnn #1 #2 #3 #4 #5
{\seq_gput_right:Nx \g_@@_blocks_seq
 \l_tmpa_tl \{ \exp_not:n { #3 } \}
 { \exp_not:n { #4 #5 } }
}
\}
```

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).

```latex
\keys_define:nn { NiceMatrix / Block / SecondPass }
{\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 ,
 \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
```

172
Some keys have not a property .value_required:n (or similar) because they are in FirstPass.

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.

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

\int_zero_new:N \l_@@_last_row_int
\int_zero_new:N \l_@@_last_col_int
We remind that the first mandatory argument of the command \texttt{Block} is the size of the block with the special format \texttt{i\textendash{}j}. However, the user is allowed to omit \texttt{i} or \texttt{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 \texttt{\_\_\_blocks_seq} as a number of rows (resp. columns) for the block equal to 100. That’s what we detect now.

\begin{verbatim}
\int_compare:nNnTF { #3 } > { 99 }
{ \int_set_eq:NN \l_@@_last_row_int \c@iRow }
{ \int_set:Nn \l_@@_last_row_int { #3 } }
\int_compare:nNnTF { #4 } > { 99 }
{ \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
{ \int_compare:nNnTF \l_@@_last_row_int > \g_@@_row_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 }
\@@_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
{ ! \l_@@_ampersand_bool }
{ \tl_gput_right:Nx \g_nicematrix_code_after_tl
{ \@@_Block_v:nnnnnn { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } }
}
\end{verbatim}

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

\begin{verbatim}
\cs_new_protected:Npn \_\_\_Block_v:nnnnnn #1 #2 #3 #4 #5 #6
{ \group_begin:
{ \int_compare:nNnTF \l_@@_last_row_int > \g_@@_row_total_int }
{ \msg_error:nnnn { nicematrix } { Block-too-large-1 } { #1 } { #2 }
\@@_Block_v:nnnnnn { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } }
\end{verbatim}

The group is for the keys.

\begin{verbatim}
\group_begin:
\int_compare:nNnT { #1 } = { #3 }
{ \str_set:Nn \l_@@_vpos_block_str { t } }
\keys_set:nn { NiceMatrix / Block / SecondPass } { #5 }
\end{verbatim}

If the content of the block contains \\	exttt{&}, we will have a special treatement (since the cell must be divided in several sub-cells).

\begin{verbatim}
\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:Nx \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
{ }
\end{verbatim}
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 \texttt{\multicolumn} and the \texttt{\diagbox} in that sequence).

\begin{verbatim}
\seq_gput_left:Nx \g_@@_pos_of_blocks_seq
{ { #1 } { #2 } { #3 } { #4 } { \l_@@_block_name_str } }
\end{verbatim}

\begin{verbatim}
\clist_if_empty:NF \l_@@_borders_clist
{ \tl_gput_right:Nx \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 }
}
}
\end{verbatim}

#5 are the options

\begin{verbatim}
\clist_if_empty:NF \l_@@_fill_tl
{ \tl_if_empty:NF \l_@@_opacity_tl
{ \tl_if_head_eq_meaning:nNTF \l_@@_fill_tl \[
{ \tl_set:Nx \l_@@_fill_tl
{ \[ opacity = \l_@@_opacity_tl ,
\tl_tail:o \l_@@_fill_tl }
}
{ \tl_set:Nx \l_@@_fill_tl
{ \{ opacity = \l_@@_opacity_tl \} \l_@@_fill_tl }
}
}
\end{verbatim}
Let’s consider the following \verb+\NiceTabular+. Because of the instruction \verb+!{\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 

\begin{pgfpicture}
\pgfrememberpicturepositiononpagetrue
\@@_qpoint:n { row - #1 } \end{pgfpicture}

We highlight the node 1-1-block-short

The construction of the node corresponding to the merged cells.
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 } \}
\str_if_empty:NF \l_@@_name_str
\{ \pgfnodealias { \l_@@_name_str - \l_@@_block_name_str } \}
\}
```

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:nnnn \l_@@_first_row_int \g_@@_row_total_int
\{ \cs_if_exist:cT
\{ \pgf @ sh @ ns \@@_env: - ##1 - #2 \}
\seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }{ west }
\dim_set:Nn \l_tmpb_dim \{ \dim_min:nn \l_tmpb_dim \pgf@x \}
\}
```

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 }{ 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:nnnn \l_@@_first_row_int \g_@@_row_total_int
\{ \cs_if_exist:cT
\}`
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.
\begin{tikzpicture}
\pgfmathsetmacro{\l_tmpa_dim}{\textwidth - \paperwidth / 2}
\pgfmathsetmacro{\l_tmpb_dim}{\ht\strutbox}
\pgfsetlinewidth{1.1 \arrayrulewidth}
\pgfsetrectcap
\pgfusepathqstroke
\end{tikzpicture}

When the final user has used the key \texttt{p}, we have to compute the width.

\begin{tikzpicture}
\pgfmathsetmacro{\l_tmpa_dim}{\textwidth - \paperwidth / 2}
\pgfmathsetmacro{\l_tmpb_dim}{\ht\strutbox}
\pgfsetlinewidth{1.1 \arrayrulewidth}
\pgfsetrectcap
\pgfusepathqstroke
\end{tikzpicture}
Now, we will put the label of the block.

\pgfrememberpicturepositiononpagetrue
\pgf@relevantforpicturesizefalse
\bool_lazy_any:NnTF { \str_if_eq_p:on \l_@@_vpos_block_str { c } \str_if_eq_p:on \l_@@_vpos_block_str { T } \str_if_eq_p:on \l_@@_vpos_block_str { B } } { }
\tl_set:Nx \l_tmpa_tl { \str_case:on \l_@@_vpos_block_str { c \centering r \raggedleft l \raggedright j } { } } #6

If we are in the first column, we must put the block as if it was with the key r.
\int_if_zero:nNnT { \int_eval:n { \l_@@_last_col_int + 1 } } { }

If we are in the last column, we must put the block as if it was with the key l.
\bool_if:nNnT { \l_@@_last_col_found_bool } { }
\int_compare:nNnT { \int_eval:n { \l_@@_col_total_int } } { }
\l_tmpa_tl will contain the anchor of the PGF node which will be used.
\tl_set:Nx \l_tmpa_tl { }
\str_case:on \l_@@_vpos_block_str { c \centering r \raggedleft l \raggedright j } { }

180
\{ 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-xsep = \c_zero_dim , \\
    inner-ysep = \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:onTF \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 \\
}
We put the label of the block which has been composed in $\text{l}@@\text{cell\_box}$.

\begin{verbatim}
\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
\endgroup
\end{verbatim}

The first argument of $\text{l}@@\text{stroke\_block}:nnn$ 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).

\begin{verbatim}
\cs_new_protected:Npn \l@@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
    \pgfreemberpicturepositiononpagetrue
    \pgf@relevantforpicturesizefalse
    \tl_if_empty:NF \l@@draw_tl
      { \CT@arc@ }
      { \@@_color:o \l@@draw_tl }
    \pgfsetcornersarced
    \pgfsetcornersarced
    { \l@@rounded_corners_dim }
    { \l@@rounded_corners_dim }
    \@@_cut_on_hyphen:w #2 \q_stop
    \int_compare:nNnF \l_tmpa_tl > \c@iRow
      { \@@_qpoint:n { row - \l_tmaa_tl } \dim_set_eq:NN \l_tmpb_dim \pgf@y
        \l@@draw_tl \l@@draw_tl \l@@draw_tl \l@@draw_tl
      }
    \endgroup
  }
\end{verbatim}

If the user has used the key color of the command $\text{Block}$ without value, the color fixed by $\text{arrayrulecolor}$ is used.

\begin{verbatim}
\l@if_eq:NTF \l@@draw_tl \c@@default_tl
  { \CT@arc@0 }
  { \@@_color:o \l@@draw_tl }
\pgfsetcornersarced
\pgfsetcornersarced
{ \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
      \l@@draw_tl \l@@draw_tl \l@@draw_tl \l@@draw_tl
    }
  }
\end{verbatim}
Here is the set of keys for the command `\@ stroke_block:nnn`.

```
\keys_define:nnn { NiceMatrix / BlockStroke }
  { color .tl_set:N = \l_@@_draw_tl ,
    draw .code:n =
    \exp_arg:N \tl_if_empty:nF { #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
  \dim_set_eq:NN \l_@@ line_width_dim \arrayrulewidth
  \keys_set_known:nn { NiceMatrix / BlockBorders } { #1 }
  \@@_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
  \int_step_inline:nnn \l_@@_tmpd_tl \l_tmpb_tl
    { \use:e
      { \@@_vline:n
        \{ position = ##1 ,
        start = \l_@@_tmpc_tl ,
        end = \l_int_eval:n { \l_tampa_tl + 1 } ,
        total-width = \dim_use:N \l_@@_line_width_dim
        \}
      }
    }
  \}
```

183
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).
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 \texttt{col} node).

\begin{verbatim}
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 \l_tmpb_dim \pgf@y } \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \pgf@y } \pgfusepathqstroke } 
  { \use:e { \exp_not:N \draw \l_@@_borders_tikz_tl } ( \pgf@x , \l_tmpb_dim ) -- ( \pgf@x , \l_@@_tmpc_dim ) ; } 
} 
\end{verbatim}

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 \texttt{row} node).

\begin{verbatim}
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_tmpb_dim { \pgf@x + 0.5 \l_@@_line_width_dim } \@@_qpoint:n \l_tampa_tl \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \l_@@_line_width_dim } \@@_qpoint:n { #1 } \tl_if_empty:NTF \l_@@_borders_tikz_tl 
  { \pgfpathmoveto { \pgfpoint \l_tampa_dim \pgf@y } \pgfpathlineto { \pgfpoint \l_@@_tmpd_dim \pgf@y } \pgfusepathqstroke } 
  { \use:e { \exp_not:N \draw \l_@@_borders_tikz_tl } \pgfusepathqstroke } 
} 
\end{verbatim}
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. The arguments #1 and #2 are the coordinates of the first cell and #3 and #4 the coordinates of the last cell of the block. #5 is a comma-separated list of the Tikz keys used with the path. However, among those keys, you have added in nicematrix a special key offset (an offset for the rectangle of the block). That’s why we have to extract that key first.
\cs_new_protected:Npn \@@_block_tikz:nnnnn #1 #2 #3 #4 #5 \{ \begin { tikzpicture } \@@_clip_with_rounded_corners: \clist_map_inline:nn { #5 } { \keys_set_known:nnN { NiceMatrix / SpecialOffset } { ##1 } \l_tmpa_tl \use:e { \exp_not:N \path \[ \l_tmpa_tl \] \( \l_tmpa_dim , \pgf@y \) -- ( \l_tmpb_dim , \pgf@y ) ; } } \end { tikzpicture } \}
\cs_generate_variant:Nn \@@_block_tikz:nnnnn { n n n n V }
\keys_define:nn { NiceMatrix / SpecialOffset } { offset .dim_set:N = \l_@@_offset_dim }

28 How to draw the dotted lines transparently

\cs_set_protected:Npn \@@_renew_matrix: \{ \RenewDocumentEnvironment { pmatrix } \{ \pNiceMatrix \endpNiceMatrix \} \RenewDocumentEnvironment { vmatrix } \{ \vNiceMatrix \endvNiceMatrix \} \}
\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
{\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 \l_@@_last_col_int > { -1 } { & } \\}
\prg_replicate:nn { #3 }
{
\int_if_zero:nT \l_@@_first_col_int { & }
}
\prg_replicate:nn { #4 - 1 } { ( } { ) #5 & } #5
\int_compare:nNnT \l_@@_last_col_int \l_@@_last_col_int > { -1 } { & } \\}
\end group
\end{NiceArrayWithDelims}
\end{NiceMatrix}
\endgroup}

The group is for the protection of the keys.

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
{\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 \l_@@_last_col_int > { -1 } { & } \\}
\prg_replicate:nn { #3 }
{
\int_if_zero:nT \l_@@_first_col_int { & }
}
\prg_replicate:nn { #4 - 1 } { ( } { ) #5 & } #5
\int_compare:nNnT \l_@@_last_col_int \l_@@_last_col_int > { -1 } { & } \\}
\end group
\end{NiceArrayWithDelims}
\end{NiceMatrix}
\endgroup}

We put \{ \} before \#6 to avoid a hasty expansion of a potential \texttt{\arabic{\textbar iRow}} at the beginning of the row which would result in an incorrect value of that \texttt{iRow} (since \texttt{iRow} is incremented in the first cell of the row of the \texttt{\halign}).
We define also a command \AutoNiceMatrix similar to the environment \{NiceMatrix\}.

30 The redefinition of the command \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}).

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

31 The command \diagbox

The command \diagbox will be linked to \diagbox nn in the environments of nicematrix. However, there are also redefinitions of \diagbox in other circumstances.
The command \@if_row_less_than:nn \{ number \} \{ instructions \}
The package \texttt{nicematrix} uses it even if \texttt{colortbl} is not loaded.

The command \texttt{\CT@arc@} is a command of \texttt{colortbl} which sets the color of the rules in the array. The package \texttt{nicematrix} uses it even if \texttt{colortbl} is not loaded.
32 The keyword \CodeAfter

In fact, in this subsection, we define the user command \CodeAfter for the case of the “normal syntax”. For the case of “light-syntax”, see the definition of the environment \{@@-light-syntax\} on p. 83.

In the environments of nicematrix, \CodeAfter will be linked to \@@_CodeAfter:. That macro must not be protected since it begins with \omit.

\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 \g_nicematrix_code_after_tl { #1 }
\@@_CodeAfter_iv:n
\}

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

\cs_new_protected:Npn \@@_CodeAfter_ii:n #1 \end
\{ 
\str_if_eq:eeTF \currenvir \#1 
\{ \end \{ \#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
\}

}
33 The delimiters in the preamble

The command \@@_delimiter:nnn will be used to draw delimiters inside the matrix when delimiters are specified in the preamble of the array. It does not concern the exterior delimiters added by \{NiceArrayWithDelims\} (and \{pNiceArray\}, \{pNiceMatrix\}, etc.).

A delimiter in the preamble of the array will write an instruction \@@_delimiter:nnn in the \g_@@_pre_code_after_tl (and also potentially add instructions in the preamble provided to \array in order to add space between columns).

The first argument is the type of delimiter (\((), \[\), \{\) or \}). The second argument is the number of column. The third argument is a boolean equal to \c_true_bool (resp. \c_false_true) when the delimiter must be put on the left (resp. right) side.

A \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
\c_math_toggle_token
\@@_color:o \l_@@_delimiters_color_tl
\bool_if:nTF { #3 } { \left #1 } { \left . } }
34 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_match:nnTF { \A[A-Za-z][A-Za-z0-9]*\Z } { #1 } \{ #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 { } }
\{ \peek_remove_spaces:n \\
\{ \tl_gput_right:Nx \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 }
\}
\NewDocumentCommand \@@_SubMatrix_in_code_before_i { > { \SplitArgument { 1 } { - } } m > { \SplitArgument { 1 } { - } } m }
\cs_new_protected:Npn \@@_SubMatrix_in_code_before_i:nnnn \@ #1 #2 #3 #4
\{ \@@_SubMatrix_in_code_before_i:nnnn \@ \#1 \#2 \#3 \#4
\}
\seq_gput_right:Nn \g_@@_submatrix_seq
\}
We use \str_if_eq:nnTF because it is fully expandable.
\{ \str_if_eq:nnTF { #1 } { last } \{ \int_use:N \c@iRow } { #1 } \}
\{ \str_if_eq:nnTF { #2 } { last } \{ \int_use:N \c@jCol } { #2 } \}
\{ \str_if_eq:nnTF { #3 } { last } \{ \int_use:N \c@iRow } { #3 } \}
\{ \str_if_eq:nnTF { #4 } { last } \{ \int_use:N \c@jCol } { #4 } \}
\}

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.

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).

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

\group_begin:

\endgroup
The last value of $\int_{step\_inline:nnn}$ is provided by currification.

#1 is the left delimiter, #2 is the right one, #3 is the subscript and #4 is the superscript.
\cs_if_exist:cT\{ pgf @ sh @ ns @ \@@_env: - \l_@@_last_i_tl - ##1 \}
{ \pgfpointanchor { \@@_env: - \l_@@_last_i_tl - ##1 } { south } \dim_set:Nn \l_@@_y_final_dim \{ \dim_min: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_CT@arc@: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 } }
{ \@@_qpoint:n { col - ##1 }
\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 } }
}
\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 \}
\pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
\pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
\pgfusepathqstroke
}
\bool_lazy_and:nnTF\{ \Wrong-line-in-SubMatrix \} { vertical } { \#1 }
}

Now, we draw the virtual 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.
\l_if_eq:NNTF \l_@@_submatrix_vlines_clist \c_@@_all_tl
{ \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 \}
\pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
\pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
\pgfusepathqstroke
}
\bool_lazy_and:nnTF\{ \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.
We use a group to protect $\l_tmpa$ and $\l_tmpb$.

\group_begin:

We compute in $\l_tmpa$ the $x$-value of the left end of the rule.

\dim_set:Nn \l_tmpa \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim
\str_case:nn { #1 } {
  \dim_sub:Nn \l_tmpa \l_tmpa \{ 0.9 \ mm \}
  \dim_sub:Nn \l_tmpa \l_tmpa \{ 0.2 \ mm \}
  \dim_sub:Nn \l_tmpa \l_tmpa \{ 0.9 \ mm \}
}\pgfpathmoveto { \pgfpoint \l_tmpa \pgf@y }

We compute in $\l_tmpb$ the $x$-value of the right end of the rule.

\dim_set:Nn \l_tmpb \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim
\str_case:nn { #2 } {
  \dim_add:Nn \l_tmpb \l_tmpb \{ 0.9 \ mm \}
  \dim_add:Nn \l_tmpb \l_tmpb \{ 0.2 \ mm \}
  \dim_add:Nn \l_tmpb \l_tmpb \{ 0.9 \ mm \}
}\pgfpathlineto { \pgfpoint \l_tmpb \pgf@y }
\pgfusepathqstroke
\group_end:

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).

\if_empty:NF \l_@@_submatrix_name_str {
  \@@_pgf_rect_node:nnnn \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 }
\pgfpoint { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
}\if_empty:NF \l_@@_submatrix_name_str {
  \@@_node_left:nn \l_@@_submatrix_name_str \#1 \{ }
}
Now, we deal with the right delimiter.

\pgftransformshift
\pgfpoint
\str_if_empty:NTF \l_@@_submatrix_name_str
\@@_node_right:nnnn #2 \l_@@_submatrix_right_xshift_dim \l_@@_y_final_dim / 2
\str_if_empty:NTF \l_@@_submatrix_name_str
\@@_node_right:nnnn #2 \l_@@_submatrix_right_xshift_dim \l_@@_y_final_dim / 2
\cs_set_eq:NN \pgfpointanchor \@@_pgfpointanchor:n
\flag_clear_new:n { nicematrix }
\l_@@_code_tl}

In the key code of the command \SubMatrix there may be Tikz instructions. We want that, in these instructions, the \textit{i} and \textit{j} in specifications of nodes of the forms \textit{i-j}, \textit{row-i}, \textit{col-j} and \textit{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_new_protected:Npn \@@_pgfpointanchor:n #1 #2
\use:e { \exp_not:N \@@_old_pgfpointanchor { \@@_pgfpointanchor_i:nn #1 } }

In fact, the argument of \pgfpointanchor is always of the form \texttt{\textbackslash a\_command \{ name\_of\_node \}} where “name\_of\_node” is the name of the Tikz node without the potential prefix and suffix. That’s why we catch two arguments and work only on the second by trying (first) to extract an hyphen -.

\cs_new:Npn \@@_pgfpointanchor_i:nn #1 #2
{ #1 \l_@@_pgfpointanchor_i:ii:w #2 - \q_stop }

Since \seq_if_in:NnTF and \clist_if_in:NnTF are not expandable, we will use the following token list and \str_case:nVTF to test whether we have an integer or not.

\tl_const:Nn \c_@@_integers_alist_tl
\{ \}
\str_case:nVTF { 1 } { 2 } { 3 } { 4 } { 5 } { 6 } { 7 } { 8 } { 9 } { 10 } { 11 } { 12 } { 13 } { 14 } { 15 } { 16 } { 17 } { 18 } { 19 } { 20 } { }

\cs_new:Npn \@@_pgfpointanchor_i:ii:w #1-#2\q_stop
If there is no hyphen, that means that the node is of the form of a single number (ex.: 5 or 11). In that case, we are in an analysis which result from a specification of node of the form $i-j$. In that case, the $i$ of the number of row arrives first (and alone) in a $\texttt{\pgfpointanchor}$ and, the, the $j$ arrives (alone) in the following $\texttt{\pgfpointanchor}$. 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 $\texttt{nicematrix}$.

```
\tl_if_empty:nTF { #2 }
{ \str_case:nVTF { #1 } \c_@@_integers_alist_tl
{ \flag_raise:n { nicematrix }
\int_if_even:nTF { \flag_height:n { nicematrix } } { \int_eval:n { #1 + \l_@@_first_i_tl - 1 } }
{ \int_eval:n { #1 + \l_@@_first_j_tl - 1 } }
}
{ #1 }
}
```

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

```
\@@_pgfpointanchor_iii:w { #1 } #2
```

There was an hyphen in the name of the node and that’s why we have to retrieve the extra hyphen we have put (cf. $\texttt{\@@_pgfpointanchor_i:nn}$).

```
\cs_new:Npn \@@_pgfpointanchor_iii:w #1 #2 -
{ \str_case:nnF { #1 } { row } { row - \int_eval:n { #2 + \l_@@_first_i_tl - 1 } }
{ col } { col - \int_eval:n { #2 + \l_@@_first_j_tl - 1 } }
}
```

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

```
\int_eval:n { #1 + \l_@@_first_i_tl - 1 }
\int_eval:n { #2 + \l_@@_first_j_tl - 1 }
```

The command $\texttt{\@@_node_left:nn}$ 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 $\texttt{\SubMatrix}$).

```
\cs_new_protected:Npn \@@_node_left:nn #1 #2
{ \pgfnode
  \{ rectangle \}
  \{ east \}
  \{ \nullfont \c_math_toggle_token \@@_color:o \l_@@_delimiters_color_tl \}
  \left #1 \vcenter {
  \nullfont \hrule \@height \l_tmpa_dim
  \@depth \c_zero_dim
  \@width \c_zero_dim
  \right .\c_math_toggle_token
}
{ #2 }
```

199
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 \c_math_toggle_token \@@_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 } } } ^ { \smash { #4 } } \c_math_toggle_token }{ #2 }{ }\}

35 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 { } } { \peek_remove_spaces:n{ \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { under } }\}
\NewDocumentCommand \@@_OverBrace { O { } m m m O { } } { \peek_remove_spaces:n{ \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { over } }\}
\keys_define:nn { NiceMatrix / Brace }{ left-shorten .bool_set:N = \l_@@_brace_left_shorten_bool , left-shorten .default:n = true , right-shorten .bool_set:N = \l_@@_brace_right_shorten_bool , shorten .meta:n = { left-shorten , right-shorten } , right-shorten .default: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_tmapa_tl , color .value_required:n = true , }
unknown .code:n = \@_error:n { 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:nTF
{ \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:nnTF { #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_set:Nn \l_@@_x_initial_dim
\dim_min:nn \l_@@_x_initial_dim \pgf@x }

}
\bool_lazy_or:nnTF
{ \bool_not_p:n \l_@@_brace_left_shorten_bool }
{ \dim_set_eq:NN \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_set:Nn \l_@@_x_final_dim
\dim_max:nn \l_@@_x_final_dim \pgf@x }

}
\bool_lazy_or:nnTF
{ \bool_not_p:n \l_@@_brace_right_shorten_bool }
{ \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_@@_last_j_tl }
{ \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east } \dim_set:Nn \l_@@_x_final_dim
\dim_max:nn \l_@@_x_final_dim \pgf@x }

}
\bool_lazy_or:nnTF
{ \bool_not_p:n \l_@@_brace_right_shorten_bool }
201
The argument is the text to put above the brace.
\cs_new_protected:Npn \@@_overbrace_i:n #1
\{ 
\@@_qpoint:n { col - \int_eval:n { \l_@@_last_i_tl + 1 } } 
\dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_final_dim \pgf@x 
\pgfset { inner-sep = \c_zero_dim } 
\str_if_eq:nnTF { #5 } { under } { \@@_underbrace_i:n { #3 } } 
{ \@@_overbrace_i:n { #3 } } 
\endpgfpicture 
\group_end: 
\}

The argument is the text to put under the brace.
\cs_new_protected:Npn \@@_underbrace_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 
\@@_math_toggle: #1 \@@_math_toggle: \cr 
\noalign { \skip_vertical:n { 3 pt } \nointerlineskip } 
\c_math_toggle_token 
\overbrace 
\{ 
\hbox_to_wd:nn 
{ \l_@@_x_final_dim - \l_@@_x_initial_dim } 
\} 
\c_math_toggle_token 
\cr 
\} 
\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 
\@@_math_toggle: #1 \@@_math_toggle: \cr 
\noalign { \skip_vertical:n { 3 pt } \nointerlineskip } 
\c_math_toggle_token 
\overbrace 
\{ 
\hbox_to_wd:nn 
{ \l_@@_x_final_dim - \l_@@_x_initial_dim } 
\} 
\c_math_toggle_token 
\cr 
\} 
\group_end: 
} 
\}

The argument is the text to put under the brace.
36 The command TikzEveryCell

```latex
\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:nTF \l_@@_in_code_after_bool \g_@@_recreate_cell_nodes_bool \{ \bool_set_true:N \l_@@_not_empty_bool \} , not-empty .value_forbidden:n = true , empty .code:n = \bool_lazy_or:nTF \l_@@_in_code_after_bool \g_@@_recreate_cell_nodes_bool \{ \bool_set_true:N \l_@@_empty_bool \} , empty .value_forbidden:n = true , unknown .code:n = \@@_error:n { Unknown-key-for-TikzEveryCell } }
\NewDocumentCommand { \@@_TikzEveryCell } { O { } m } { \IfPackageLoadedTF { tikz } { } { } }
```

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 } }

\tl_new:N \@@_i_tl
\tl_new:N \@@_j_tl
\cs_new_protected:Nn \@@_all_the_cells:
{\int_step_variable:nNn { \int_use:c { c@iRow } } \@@_i_tl
{\int_step_variable:nNn { \int_use:c { c@jCol } } \@@_j_tl
 { \cs_if_exist:cF { cell - \@@_i_tl - \@@_j_tl }
   { \exp_args:NNe \seq_if_in:NnF \l_@@_corners_cells_seq
     { \@@_i_tl - \@@_j_tl }
     { \bool_set_false:N \l_tmpa_bool
       \cs_if_exist:cTF
       { pgf @ sh @ ns @ \@@_env: - \@@_i_tl - \@@_j_tl } }
     { \bool_if:NT \l_@@_empty_bool
       { \bool_set_true:N \l_tmpa_bool }
     }
     { \bool_if:NT \l_@@_not_empty_bool
       { \bool_set_true:N \l_tmpa_bool }
     }
   }
\bool_if:NF \l_@@_empty_bool
 { \@@_mark_cells_of_block:nnnn { #1 } { #2 } { #3 } { #4 } \l_tmpa_tl
 }
\cs_new_protected:Nn \@@_for_a_block:nnnnn
{ \bool_if:NF \l_@@_empty_bool
 { \@@_block_tikz:nnnnV \@@_i_tl \@@_j_tl \@@_i_tl \@@_j_tl \l_tmpa_tl
 }
\cs_new_protected:Nn \@@_mark_cells_of_block:nnnn
{ \int_step_inline:nnn { #1 } #3
  \@@_mark_cells_of_block:nnnn { #1 } #2
}
\cs_new_protected:Nn \@@_mark_cells_of_block:nnnn
{ \int_step_inline:n { #1 } { #3 }

204
37 The command \ShowCellNames

\NewDocumentCommand \@@_ShowCellNames_CodeBefore { } \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 { \begin { pgfpicture } \@@_qpoint:n { row - ##1 } \dim_set_eq:NN \l_tmpa_dim \pgf@y \@@_qpoint:n { row - \int_eval:n { ##1 + 1 } } \dim_set_eq:NN \g_tmpa_dim { ( \l_tmpa_dim + \pgf@y ) / 2 } \dim_set_eq:NN \g_tmpb_dim { \l_tmpa_dim - \pgf@y } \bool_if:NTF \l_@@_in_code_after_bool \end { pgfpicture } \int_step_inline:nn \c@jCol { \hbox_set:Nn \l_tmpa_box { \normalfont \Large \color { red ! 50 } ##1 - ####1 } \begin { pgfpicture } \@@_qpoint:n { col - ####1 } \dim_set_eq:NN \g_@@_tmpc_dim \pgf@x \@@_qpoint:n { col - \int_eval:n { ####1 + 1 } } \dim_set_eq:NN \g_@@_tmpe_dim \pgf@x \dim_set_eq:NN \g_@@_tmpd_dim { \pgf@x - \g_@@_tmpc_dim } \bool_if:NTF \l_@@_in_code_after_bool \end { pgfpicture } \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_@@_tmpe_dim } \box_ht_plus_dp:N \l_tmpa_box } } \box_scale:Nnn \l_tmpa_box \fp_use:N \l_tmpa_fp \fp_use:N \l_tmpa_fp \pgfpicture \pgfrememberpictureonpagetrue \pgf@relevantforpicturesizefalse \pgftransformshift \box_scale:Nnn \l_tmpa_box \fp_use:N \l_tmpa_fp \fp_use:N \l_tmpa_fp \pgfpicture \pgfstopmemorizedpicturepositionnonpagetrue \pgf@relevantforpicturesizefalse \pgftransformshift
}
38 We process the options at package loading

We process the options when the package is loaded (with \usepackage) but we recommend to use \NiceMatrixOptions instead.

We must process these options after the definition of the environment \{NiceMatrix\} because the option \texttt{renew-matrix} executes the code \texttt{\cs_set_eq:NN \env@matrix \NiceMatrix}.

Of course, the command \texttt{\NiceMatrix} must be defined before such an instruction is executed.

The boolean \texttt{\g_@@_footnotehyper_bool} will indicate if the option \texttt{footnotehyper} is used.

The boolean \texttt{\g_@@_footnote_bool} will indicate if the option \texttt{footnote} is used, but quickly, it will also be set to \texttt{true} if the option \texttt{footnotehyper} is used.

\begin{verbatim}
\bool_new:N \g_@@_footnotehyper_bool
\msg_new:nnnn { nicematrix } { Unknown~key~for~package }
\{ \The-key-\l_keys_key_str-is-unknown. \}
\end{verbatim}
That key will be ignored. \\
For a list of the available keys, type H <return>.

\keys_define:nn { NiceMatrix / Package }
\{ 
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 ,
no-test-for-array .bool_set:N = \g_@@_no_test_for_array_bool ,
no-test-for-array .default:n = true ,
unknown .code:n = \@@_error:n { Unknown key for package }
\}
\ProcessKeysOptions { NiceMatrix / Package }
\@@_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
\{ 

208
The class \texttt{beamer} has its own system to extract footnotes and that’s why we have nothing to do if \texttt{beamer} is used.

\begin{verbatim}
\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
\end{verbatim}

The flag \texttt{\g_@@_footnote_bool} is raised and so, we will only have to test \texttt{\g_@@_footnote_bool} in order to know if we have to insert an environment \texttt{savenotes}.

\section*{39 About the package underscore}

If the user loads the package \texttt{underscore}, it must be loaded \textit{before} the package \texttt{nicematrix}. If it is loaded after, we raise an error.

\begin{verbatim}
\bool_new:N \l_@@_underscore_loaded_bool
\IfPackageLoadedTF { underscore }
  { \bool_set_true:N \l_@@_underscore_loaded_bool }
\hook_gput_code:nnn { begindocument } { . }
\bool_if:NF \l_@@_underscore_loaded_bool
  { \IfPackageLoadedTF { underscore }
    { \@@_error:n { underscore-after-nicematrix } }
    { }
  }\end{verbatim}

\section*{40 Error messages of the package}

\begin{verbatim}
\bool_if:NTF \g_@@_messages_for_Overleaf_bool
  { \str_const:Nn \c_@@_available_keys_str { } }
  { \str_const:Nn \c_@@_available_keys_str
    { 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_x:NNn \g_@@_types_of_matrix_seq \g_@@_types_of_matrix_seq
  { \tl_to_str:n { #1 } }
\end{verbatim}

If the user uses too much columns, the command \texttt{\@@_error_too_much_cols}: is triggered. This command raises an error but also tries to give the best information to the user in the error message.
The command \seq_if_in:NoTF is not expandable and that's why we can't put it in the error message itself. We have to do the test before the \@@_fatal:n.

\cs_new_protected:Npn \@@_error_too_much_cols:
\seq_if_in:NoTF \g_@@_types_of_matrix_seq \g_@@_name_env_str
\int_compare:nNnTF \l_@@_last_col_int = { -2 }
{ \@@_fatal:n { too-much-cols-for-matrix } }
{ }
\int_compare:nNnTF \l_@@_last_col_int = { -1 }
{ \@@_fatal:n { too-much-cols-for-matrix } }
{ }
\bool_if:NF \l_@@_last_col_without_value_bool
{ \@@_fatal:n { too-much-cols-for-matrix-with-last-col } }
{ \@@_fatal:nn { too-much-cols-for-array } }
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\texttt{\Hdotsfor}~is~incorrect. }
\@@_msg_new:nn { hvlines,-rounded-corners-and-corners }
{ Incompatibile-options. \\ }
{ You-should-not-use~'hvlines',~'-rounded-corners'~and~'corners'~at~this~time. \\ }
{ The-output~will-not-be-reliable. }
\@@_msg_new:nn { negative-weight }
{ Negative-weight. \\ }
{ The-weight-of-the~'X'~-columns-must-be-positive-and-you-have-used~
the-value-'\int_use:N \l_@@_weight_int'. \\ }
{ The-absolute-value-will-be-used. }
\@@_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\texttt{\@@_full_name_env}:~However,~you-can-go-on. }
\@@_msg_new:nn { too-much-cols-for-matrix-with-last-col }
{ Too-much-columns. \\ }
{ In-the-row\texttt{\int_eval:n \{ \c@iRow \}}~,-
you-try-to-use-more-columns~
than-allowed-by-your\texttt{\@@_full_name_env}:\@@_message_hdotsfor:~
The-maximal-number-of-columns-is\texttt{\int_eval:n \{ \l_@@_last_col_int - 1 \}}~
(plus-the-exterior-columns).~This-error-is-fatal. }
\@@_msg_new:nn { too-much-cols-for-matrix }
{ Too-much-columns. \\ }
{ In-the-row\texttt{\int_eval:n \{ \c@iRow \}}~,-
you-try-to-use-more-columns-than-allowed-by-your~
\texttt{\@@_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-much-cols-for-array }
{
Too much columns.\%\nIn-the-row-\int_eval:n { \c@iRow },-you-try-to-use-more-columns-than-allowed-by-your-\\_full_name_env::\_message_dotsfor:\ The-maximal-number-of-columns-is-\int_use:N \g_@@_static_num_of_col_int-
-(plus-the-potential-exterior-ones).
This error is fatal.
}

\@@_msg_new:nn { columns-not-used }
{
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.-The-columns-you-did-not-used-won't-be-created.\%
You won't have similar error message 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 \texttt{#1} in the first column (number 0) of the array.\%
That command will be ignored.
}

\@@_msg_new:nn { in-last-col }
{
Erroneous-use.\%
You can't use the command \texttt{#1} in the last column (exterior) of the array.\%
That command will be ignored.
}

\@@_msg_new:nn { in-first-row }
{
Erroneous-use.\%
You can't use the command \texttt{#1} in the first row (number 0) of the array.\%
That command will be ignored.
}

\@@_msg_new:nn { in-last-row }
{
Erroneous-use.\%
You can't use the command \texttt{#1} in the last row (exterior) of the array.\%
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. 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'.

```latex
\@@_msg_new:nn { double-closing-delimiter }
{ Double-delimiter.\} 
\@@_msg_new:nn { delimiter-after-opening }
{ Double-delimiter.\} 
\@@_msg_new:nn { bad-option-for-line-style }
{ Bad-line-style.\} 
\@@_msg_new:nn { Identical-notes-in-caption }
{ Identical-tabular-notes.\} 
\@@_msg_new:nn { tabularnote-below-the-tabular }
{ \token_to_str:N \tabularnote\ forbidden\} 
\@@_msg_new:nn { Unknown-key-for-rules }
{ Unknown-key.\} 
\@@_msg_new:nn { Unknown-key-for-TikzEveryCell }
{ Unknown-key.\} 
\@@_msg_new:nn { Unknown-key-for-rotate }
{ Unknown-key.\} 
\@@_msg_new:nn { Unknown-key-for-custom-line }
{ Unknown-key.\} 
```

212
The key ‘\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):
  command, -
  color, -
  command, -
  dotted, -
  letter, -
  multiplicity, -
  sep-color, -
  tikz, and total-width.
}
\@@_msg_new:nnn { Unknown key for xdots }
{ Unknown key. \*
The key ‘\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-labels’, -
  ‘inter’, -
  ‘line-style’, -
  ‘radius’, -
  ‘shorten’, -
  ‘shorten-end’ and ‘shorten-start’.
}
\@@_msg_new:nnn { 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 ‘\keys_key_str’)
That key will be ignored.
}
\@@_msg_new:nnn { 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:nnn { W warning }
{ Line ‘\msg_line_number:’ The cell is too wide for your column ‘W’ -
(row ‘\int_use:N \c@iRow’).
}
\@@_msg_new:nnn { 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:nnn { 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 \text{~times}’.
\@@_msg_new:nn { ampersand-in-light-syntax }
{ Ampersand-forbidden.\\n  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.\\n  You can't use-\token_to_str:N N \-to-separate-rows-because-the-key-'light-syntax'-
  is-in-force.-You must-use-the-character-\token_to_str:N l_@@_end_of_row_tl-
  (set-by-the-key-'end-of-row').-This-error-is-fatal. }
\@@_msg_new:nn { hlines-with-color }
{ Incompatible-keys.\\n  You can't use the keys-'hlines','\'vlines' or-'h\vlines'-for-a-
  \token_to_str:N \Block'-when-the-key-'color'-or-'draw'-is-used.\\n  However, you can put several commands-\token_to_str:N N \Block.\\n  Your-key-will-be-discarded. }
\@@_msg_new:nn { bad-value-for-baseline }
{ Bad-value-for-baseline.\\n  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'.\\n  A value of-1 will be used. }
\@@_msg_new:nn { detection-of-empty-cells }
{ Problem with-'not-empty'\\n  For technical reasons, you must activate-
  'create-cell-nodes'-in-\token_to_str:N CodeBefore\n  in order to use the key-\token_to_str:N l_@@_keys_key_str.\\n  That key will be ignored. }
\@@_msg_new:nn { siunitx-not-loaded }
{ siunitx-not-loaded\\n  You can't use the columns-'S'-because-'siunitx'-is-not-loaded.\\n  That error is fatal. }
\@@_msg_new:nn { ragged2e-not-loaded }
{ You have to load-\token_to_str:N \SubMatrix-in-order-to-use-the-key-
  \token_to_str:N l_@@_keys_key_str'-in-
  your column-\token_to_str:N l_@@_vpos_col_str'(or-\token_to_str:N X).-The key-'\str_lowercase:o-
  \token_to_str:N l_@@_keys_key_str'-will-be-used-instead. }
\@@_msg_new:nn { Invalid-name }
{ Invalid-name.\\n  You can't give the name-\token_to_str:N \SubMatrix-of-your-\token_to_str:N \@full_name_env:-\\n  A name must be accepted by the regular expression-[A-Za-z][A-Za-z0-9]*.\\n  This key will be ignored. }

Wrong-line.

Wrong-line-in-SubMatrix

You-try-to-draw-a-line-of-number #' in-a-
\token_to_str:N \SubMatrix of-your-\@@_full_name_env: but-that-
number-is-not-valid.-It-will-be-ignored.

Impossible-delimiter.

Impossible-delimiter-in-
\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.

width-without-X-columns

You-have-used-the-key- 'width'-but-you-have-put-no- 'X'-column.-
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.

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.

empty-environment

Empty-environment.

Your-\@@_full_name_env: is-empty.-This-error-is-fatal.

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.

Forbidden-letter

Forbidden-letter.

You-can't-use-the-letter- '1'-for-a-customized-line.
It-will-be-ignored.

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\ 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).-\nThe-available-values-are:-left,-right,-top-and-bottom-(and-you-can-
also-use-the-key-'tikz'
\IfPackageLoadedTF { 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.-
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.-
This-key-will-be-ignored.
}
\@@_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-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 { tabularnote-forbidden }
\{ Forbidden-command.\}
\{ You-can't-use-the-command-\token_to_str:N\tabularnote\ -here.-This-command-is-available-only-in-
{NiceTabular},-{NiceTabular*} -or-in-the-argument-of-a-command-\token_to_str:N \caption\ included-
in-an-environment-{table}. \}\nThis-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 { without-color-inside }
\{ If-order-to-use-\token_to_str:N \cellcolor,-\token_to_str:N \rowcolor,-
outside-\token_to_str:N \CodeBefore, you should have used the key 'color-inside' in your-\@@_full_name_env:\ You can go on but you may need more compilations.
}

\@@_msg_new:nn { color-in-custom-line-with-tikz }
{ Erroneous use.\\\n 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.\\\n 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. You can avoid this problem by using 'last-row'=\\\n without value (more compilations might be necessary).\}

\@@_msg_new:nn { Yet-in-env }
{ Nested environments.\\\n Environments of nicematrix can't be nested.\ This error is fatal.\}

\@@_msg_new:nn { Outside-math-mode }
{ Outside math mode.\\\n 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.\\\n The value of key '='\l_keys_key_str' must be of length 1.\ It will be ignored.\}

\@@_msg_new:nn { TabularNote-in-CodeAfter }
{ Environment (TabularNote)-forbidden.\\\n 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.\\\n You can't use the column type 'V' because 'varwidth' is not loaded.\ Your column will behave like 'p'.\}

\@@_msg_new:nnn { Unknown-key-for-RulesBis }
{ Unknown 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):-b,-B,-borders,-c,-draw,-fill,-
  hlines,-vlines,-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)-
  - 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). 
  \} 
\}
 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'-(with-the-subkeys-"xshift"(which-sets-both-"left-xshift"-

and-"right-xshift")).\\

\}

 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,-

detect-duplicates,-

'enumitem-keys-',-

'enumitem-keys-para,-

'para,-

'label-in-list,-

'label-in-tabular-and-

'style. 

\}

 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',-

'nb-rows'-and-

'rowcolor'. 

\}
That key will be ignored.

\c__available_keys_str
{
The available keys are (in alphabetic order):
allow-duplicate-names, -
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, -
vlines, -
vlines-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 1 and r.
This error message is used for the set of keys \texttt{NiceMatrix/NiceMatrix} and \texttt{NiceMatrix/pNiceArray} (but not by \texttt{NiceMatrix/NiceArray} because, for this set of keys, there is no \texttt{l} and \texttt{r}).

\begin{verbatim}
\@_msg_new:nnn { Unknown-key-for-NiceMatrix }
\{ Unknown-key. \\
\textbackslash{}l\_keys\_key\_str'-is-unknown-for-the- \\
\@\_full\_name\_env: . \textbackslash{}l \\
That-key-will-be-ignored. \textbackslash{}l \\
\c\_\@\_available\_keys\_str
\}
\{ The-available-keys-are-(in-alphabetic-order):- \\
b,- \\
c,- \\
baseline,- \\
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,- \\
color-inside,- \\
columns-width,- \\
corners,- \\
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.
\}
\end{verbatim}
\@@_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):-b,\-c,\-caption,-cell-space-bottom-limit,-cell-space-limits,-cell-space-top-limit,-code-after,-code-for-first-col,-

\@@_msg_new:nnn { Duplicate-name }
\{
  Duplicate-name.\\n  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'.\\n  \bool_if:NF \g_@@_messages_for_Overleaf_bool
\{
  \For-a-list-of-the-names-already-used,-type-H<return>.-
}\}
\@@_msg_new:nnn { Option-auto-for-columns-width }
\{
  Erroneous-use.\\n  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. }
The command \diagbox 188
The keyword \CodeAfter 190
The delimiters in the preamble 191
The command \SubMatrix 192
Les commandes \UnderBrace et \OverBrace 200
The command TikzEveryCell 203
The command \ShowCellNames 205
We process the options at package loading 207
About the package underscore 209
Error messages of the package 209