

The package `nicematrix`*

F. Pantigny
fpantigny@wanadoo.fr

February 16, 2022

Abstract

The LaTeX package `nicematrix` provides new environments similar to the classical environments `{tabular}`, `{array}` and `{matrix}` of `array` and `amsmath` but with extended features.

$$\begin{array}{c} L_1 \\ L_2 \\ \vdots \\ L_n \end{array} \begin{array}{c} C_1 \\ C_2 \cdots \cdots C_n \end{array} \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}$$

Product	dimensions (cm)			Price
	L	l	h	
small	3	5.5	1	30
standard	5.5	8	1.5	50.5
premium	8.5	10.5	2	80
extra	8.5	10	1.5	85.5
special	12	12	0.5	70

The package `nicematrix` is entirely contained in the file `nicematrix.sty`. This file may be put in the current directory or in a `texmf` tree. However, the best is to install `nicematrix` with a TeX distribution such as MiKTeX, TeXLive or MacTeX.

Remark: If you use LaTeX via Internet with, for example, Overleaf, you can upload the file `nicematrix.sty` in the repertory of your project in order to take full advantage of the latest version de `nicematrix`.¹

This package can be used with `xelatex`, `lualatex`, `pdflatex` but also by the classical workflow `latex-dvips-ps2pdf` (or Adobe Distiller). However, the file `nicematrix.dtx` of the present documentation should be compiled with XeLaTeX.

This package requires and **loads** the packages `l3keys2e`, `array`, `amsmath`, `pgfcore` and the module `shapes` of PGF (`tikz`, which is a layer over PGF is *not* loaded). The final user only has to load the package with `\usepackage{nicematrix}`.

If you use TeXLive as TeX distribution, you should note that TeXLive 2020 at least is required by `nicematrix`.

The idea of `nicematrix` is to create PGF nodes under the cells and the positions of the rules of the tabular created by `array` and to use these nodes to develop new features. As usual with PGF, the coordinates of these nodes are written in the `aux` to be used on the next compilation and that's why `nicematrix` may need **several compilations**.²

Most features of `nicematrix` may be used without explicit use of PGF or Tikz (which, in fact, is not loaded by default).

A command `\NiceMatrixOptions` is provided to fix the options (the scope of the options fixed by this command is the current TeX group: they are semi-global).

*This document corresponds to the version 6.6 of `nicematrix`, at the date of 2022/02/16.

¹The latest version of the file `nicematrix.sty` may be downloaded from the SVN server of TeXLive:

<https://www.tug.org/svn/texlive/trunk/Master/texmf-dist/tex/latex/nicematrix/nicematrix.sty>

²If you use Overleaf, Overleaf will do automatically the right number of compilations.

1 The environments of this package

The package `nicematrix` defines the following new environments.

<code>{NiceTabular}</code>	<code>{NiceArray}</code>	<code>{NiceMatrix}</code>
<code>{NiceTabular*}</code>	<code>{pNiceArray}</code>	<code>{pNiceMatrix}</code>
<code>{NiceTabularX}</code>	<code>{bNiceArray}</code>	<code>{bNiceMatrix}</code>
	<code>{BNiceArray}</code>	<code>{BNiceMatrix}</code>
	<code>{vNiceArray}</code>	<code>{vNiceMatrix}</code>
	<code>{VNiceArray}</code>	<code>{VNiceMatrix}</code>

The environments `{NiceArray}`, `{NiceTabular}` and `{NiceTabular*}` are similar to the environments `{array}`, `{tabular}` and `{tabular*}` of the package `array` (which is loaded by `nicematrix`).

The environments `{pNiceArray}`, `{bNiceArray}`, etc. have no equivalent in `array`.

The environments `{NiceMatrix}`, `{pNiceMatrix}`, etc. are similar to the corresponding environments of `amsmath` (which is loaded by `nicematrix`): `{matrix}`, `{pmatrix}`, etc.

The environment `{NiceTabularX}` is similar to the environment `{tabularx}` from the eponymous package.³

It's recommended to use primarily the classical environments and to use the environments of `nicematrix` only when some feature provided by these environments is used (this will save memory).

All the environments of the package `nicematrix` accept, between square brackets, an optional list of `key=value` pairs. **There must be no space before the opening bracket (`[`) of this list of options.**

2 The vertical space between the rows

It's well known that some rows of the arrays created by default with LaTeX are, by default, too close to each other. Here is a classical example.

```
\begin{pmatrix}
\frac{1}{2} & -\frac{1}{2} \\
\frac{1}{3} & \frac{1}{4} \\
\end{pmatrix}
```

Inspired by the package `cellspace` which deals with that problem, the package `nicematrix` provides two keys `cell-space-top-limit` and `cell-space-bottom-limit` similar to the parameters `\cellspacetoplimit` and `\cellspacebottomlimit` of `cellspace`.

There is also a key `cell-space-limits` to set both parameters at once.

The initial value of these parameters is 0 pt in order to have for the environments of `nicematrix` the same behaviour as those of `array` and `amsmath`. However, a value of 1 pt would probably be a good choice and we suggest to set them with `\NiceMatrixOptions`.⁴

```
\NiceMatrixOptions{cell-space-limits = 1pt}

\begin{pNiceMatrix}
\frac{1}{2} & -\frac{1}{2} \\
\frac{1}{3} & \frac{1}{4} \\
\end{pNiceMatrix}
```

³In fact, it's possible to use directly the `X` columns in the environment `{NiceTabular}` (and the required width for the tabular is fixed by the key `width`): cf. p. 21

⁴One should remark that these parameters apply also to the columns of type `S` of `siunitx` whereas the package `cellspace` is not able to act on such columns of type `S`.

3 The vertical position of the arrays

The package `nicematrix` provides a option `baseline` for the vertical position of the arrays. This option takes in as value an integer which is the number of the row on which the array will be aligned.

```
$A = \begin{pNiceMatrix}[baseline=2]
\frac{1}{\sqrt{1+p^2}} & p & 1-p \\
1 & 1 & 1 \\
1 & p & 1+p
\end{pNiceMatrix}$
```

$$A = \begin{pmatrix} \frac{1}{\sqrt{1+p^2}} & p & 1-p \\ 1 & 1 & 1 \\ 1 & p & 1+p \end{pmatrix}$$

It's also possible to use the option `baseline` with one of the special values `t`, `c` or `b`. These letters may also be used absolutely like the option of the environments `{tabular}` and `{array}` of `array`. The initial value of `baseline` is `c`.

In the following example, we use the option `t` (equivalent to `baseline=t`) immediately after an `\item` of list. One should remark that the presence of a `\hline` at the beginning of the array doesn't prevent the alignment of the baseline with the baseline of the first row (with `{tabular}` or `{array}` of `array`, one must use `\firsthline`).

```
\begin{enumerate}
\item an item
\smallskip
\item \renewcommand{\arraystretch}{1.2}
$\begin{NiceArray}[t]{lcccccc}
\hline
n & 0 & 1 & 2 & 3 & 4 & 5 \\
u_n & 1 & 2 & 4 & 8 & 16 & 32
\hline
\end{NiceArray}$
\end{enumerate}
```

1. an item

2.	n	0	1	2	3	4	5
	u_n	1	2	4	8	16	32

However, it's also possible to use the tools of `booktabs`⁵: `\toprule`, `\bottomrule`, `\midrule`, etc.

```
\begin{enumerate}
\item an item
\smallskip
\item
$\begin{NiceArray}[t]{lcccccc}
\toprule
n & 0 & 1 & 2 & 3 & 4 & 5 \\
\midrule
u_n & 1 & 2 & 4 & 8 & 16 & 32
\bottomrule
\end{NiceArray}$
\end{enumerate}
```

1. an item

2.	n	0	1	2	3	4	5
	u_n	1	2	4	8	16	32

It's also possible to use the key `baseline` to align a matrix on an horizontal rule (drawn by `\hline`). In this aim, one should give the value `line-i` where *i* is the number of the row *following* the horizontal rule.

```
\NiceMatrixOptions{cell-space-limits=1pt}

$A=\begin{pNiceArray}{cc|cc}[baseline=line-3]
\dfrac{1}{A} & \dfrac{1}{B} & 0 & 0 \\
\dfrac{1}{C} & \dfrac{1}{D} & 0 & 0 \\
\hline
0 & 0 & A & B \\
0 & 0 & D & D
\end{pNiceArray}$
```

$$A = \left(\begin{array}{cc|cc} \frac{1}{A} & \frac{1}{B} & 0 & 0 \\ \frac{1}{C} & \frac{1}{D} & 0 & 0 \\ \hline 0 & 0 & A & B \\ 0 & 0 & D & D \end{array} \right)$$

⁵The extension `booktabs` is *not* loaded by `nicematrix`.

4 The blocks

4.1 General case

In the environments of `nicematrix`, it's possible to use the command `\Block` in order to place an element in the center of a rectangle of merged cells of the array.⁶

The command `\Block` must be used in the upper leftmost cell of the array with two arguments.

- The first argument is the size of the block with the syntax i - j where i is the number of rows of the block and j its number of columns.

If this argument is empty, its default value is 1-1. If the number of rows is not specified, or equal to *, the block extends until the last row (idem for the columns).

- The second argument is the content of the block. It's possible to use `\\` in that content to have a content on several lines. In `{NiceTabular}`, `{NiceTabular*}` and `{NiceTabularX}`, the content of the block is composed in text mode whereas, in the other environments, it is composed in math mode.

Here is an example of utilisation of the command `\Block` in mathematical matrices.

```
$\begin{bNiceArray}{cw{c}{1cm}c|c}[margin]
\Block{3-3}{A} & & 0 \\
& & \Vdots \\
& & 0 \\
\hline
0 & \Cdots & 0 & 0 \\
\end{bNiceArray}$
```

$$\left[\begin{array}{c|c} A & \begin{smallmatrix} 0 \\ \vdots \\ 0 \end{smallmatrix} \\ \hline 0 \cdots \cdots 0 & 0 \end{array} \right]$$

One may wish to raise the size of the “A” placed in the block of the previous example. Since this element is composed in math mode, it's not possible to use directly a command like `\large`, `\Large` and `\LARGE`. That's why the command `\Block` provides an option between angle brackets to specify some TeX code which will be inserted before the beginning of the math mode.⁷

```
$\begin{bNiceArray}{cw{c}{1cm}c|c}[margin]
\Block{3-3}<\Large>{A} & & 0 \\
0 & & \Vdots \\
& & 0 \\
\hline
0 & \Cdots & 0 & 0 \\
\end{bNiceArray}$
```

$$\left[\begin{array}{c|c} A & \begin{smallmatrix} 0 \\ \vdots \\ 0 \end{smallmatrix} \\ \hline 0 \cdots \cdots 0 & 0 \end{array} \right]$$

It's possible to set the horizontal position of the block with one of the keys `l`, `c` and `r`.

```
$\begin{bNiceArray}{cw{c}{1cm}c|c}[margin]
\Block[r]{3-3}<\LARGE>{A} & & 0 \\
& & \Vdots \\
& & 0 \\
\hline
0 & \Cdots & 0 & 0 \\
\end{bNiceArray}$
```

$$\left[\begin{array}{c|c} A & \begin{smallmatrix} 0 \\ \vdots \\ 0 \end{smallmatrix} \\ \hline 0 \cdots \cdots 0 & 0 \end{array} \right]$$

In fact, the command `\Block` accepts as first optional argument (between square brackets) a list of couples `key=value`. The available keys are as follows:

⁶The spaces after a command `\Block` are deleted.

⁷This argument between angular brackets may also be used to insert a command of font such as `\bfseries` when the command `\\` is used in the content of the block.

- the keys `l`, `c` and `r` are used to fix the horizontal position of the content of the block, as explained previously;
- the key `fill` takes in as value a color and fills the block with that color;
- the key `draw` takes in as value a color and strokes the frame of the block with that color (the default value of that key is the current color of the rules of the array);
- the key `color` takes in as value a color and apply that color the content of the block but draws also the frame of the block with that color;
- the key `line-width` is the width (thickness) of the frame (this key should be used only when the key `draw` or the key `hvlines` is in force);
- the key `rounded-corners` requires rounded corners (for the frame drawn by `draw` and the shape drawn by `fill`) with a radius equal to the value of that key (the default value is 4 pt⁸);
- the key `borders` provides the ability to draw only some borders of the blocks; the value of that key is a (comma-separated) list of elements covered by `left`, `right`, `top` and `bottom`;
- the keys `t` and `b` fix the base line that will be given to the block when it has a multi-line content (the lines are separated by `\\`);
- the keys `hlines`, `vlines` and `hvlines` draw all the corresponding rules in the block;
- when the key `tikz` is used, the Tikz path corresponding of the rectangle which delimits the block is executed with Tikz⁹ by using as options the value of that key `tikz` (which must be a list of keys allowed for a Tikz path). For examples, cf. p. 47;
- **New 6.3** the key `name` provides a name to the rectangular Tikz node corresponding to the block; it's possible to use that name with Tikz in the `\CodeAfter` of the environment (cf. p. 28);
- **New 6.5** the key `respect-arraystretch` prevents the setting of `\arraystretch` to 1 at the beginning of the block (which is the behaviour by default).

One must remark that, by default, the commands `\Blocks` don't create space. There is exception only for the blocks mono-row and the blocks mono-column as explained just below.

In the following example, we have had to enlarge by hand the columns 2 and 3 (with the construction `wc{...}` of array).

```
\begin{NiceTabular}{cwc{2cm}wc{3cm}c}
rose      & tulip & daisy & dahlia \\
violet
& \Block[draw=red,fill=[RGB]{204,204,255},rounded-corners]{2-2}
& {\LARGE Some beautiful flowers}
& & marigold \\
iris & & & lis \\
arum & periwinkle & forget-me-not & hyacinth
\end{NiceTabular}
```

rose	tulip	daisy	dahlia
violet	Some beautiful flowers		marigold
iris			lis
arum	periwinkle	forget-me-not	hyacinth

⁸This value is the initial value of the *rounded corners* of Tikz.

⁹Tikz should be loaded (by default, `nicematrix` only loads PGF) and, if it's not, an error will be raised.

4.2 The mono-column blocks

The mono-column blocks have a special behaviour.

- The natural width of the contents of these blocks is taken into account for the width of the current column.
In the columns with a fixed width (columns `w{...}{...}`, `p{...}`, `b{...}`, `m{...}` and `X`), the content of the block is formatted as a paragraph of that width.
- The specification of the horizontal position provided by the type of column (`c`, `r` or `l`) is taken into account for the blocks.
- The specifications of font specified for the column by a construction `>{...}` in the preamble of the array are taken into account for the mono-column blocks of that column (this behaviour is probably expected).

```
\begin{NiceTabular}{@{}>{\bfseries}lr@{}} \hline
\Block{2-1}{John}    & 12 \\
                    & 13 \\ \hline
Steph               & 8  \\ \hline
\Block{3-1}{Sarah}   & 18 \\
                    & 17 \\
                    & 15 \\ \hline
Ashley              & 20 \\ \hline
Henry               & 14 \\ \hline
\Block{2-1}{Madison} & 15 \\
                    & 19 \\ \hline
\end{NiceTabular}
```

John	12
	13
Steph	8
	18
Sarah	17
	15
Ashley	20
Henry	14
	15
Madison	19

4.3 The mono-row blocks

For the mono-row blocks, the natural height and depth are taken into account for the height and depth of the current row (as does a standard `\multicolumn` of LaTeX).

4.4 The mono-cell blocks

A mono-cell block inherits all the properties of the mono-row blocks and mono-column blocks.

At first sight, one may think that there is no point using a mono-cell block. However, there are some good reasons to use such a block.

- It's possible to use the command `\` in a (mono-cell) block.
- It's possible to use the option of horizontal alignment of the block in derogation of the type of column given in the preamble of the array.
- It's possible to draw a frame around the cell with the key `draw` of the command `\Block` and to fill the background with rounded corners with the keys `fill` and `rounded-corners`.¹⁰
- It's possible to draw one or several borders of the cell with the key `borders`.

¹⁰If one simply wishes to color the background of a unique cell, there is no point using the command `\Block`: it's possible to use the command `\cellcolor` (when the key `colortbl-like` is used).

```

\begin{NiceTabular}{cc}
\toprule
Writer & \Block[1]{year of birth} \\
\midrule
Hugo & 1802 \\
Balzac & 1799 \\
\bottomrule
\end{NiceTabular}

```

Writer	year of birth
Hugo	1802
Balzac	1799

We recall that if the first mandatory argument of `\Block` is left blank, the block is mono-cell.¹¹

4.5 Horizontal position of the content of the block

By default, the horizontal position of the content of a block is computed by using the positions of the *contents* of the columns implied in that block. That's why, in the following example, the header “First group” is correctly centered despite the instruction `!\qquad` in the preamble which has been used to increase the space between the columns (this is not the behaviour of `\multicolumn`).

```

\begin{NiceTabular}{@{}c!\qquad ccc!\qquad ccc@{}}
\toprule
Rank & \Block{1-3}{First group} & & & \Block{1-3}{Second group} \\
& 1A & 1B & 1C & 2A & 2B & 2C \\
\midrule
1 & 0.657 & 0.913 & 0.733 & 0.830 & 0.387 & 0.893 \\
2 & 0.343 & 0.537 & 0.655 & 0.690 & 0.471 & 0.333 \\
3 & 0.783 & 0.885 & 0.015 & 0.306 & 0.643 & 0.263 \\
4 & 0.161 & 0.708 & 0.386 & 0.257 & 0.074 & 0.336 \\
\bottomrule
\end{NiceTabular}

```

Rank	First group			Second group		
	1A	1B	1C	2A	2B	2C
1	0.657	0.913	0.733	0.830	0.387	0.893
2	0.343	0.537	0.655	0.690	0.471	0.333
3	0.783	0.885	0.015	0.306	0.643	0.263
4	0.161	0.708	0.386	0.257	0.074	0.336

In order to have an horizontal positioning of the content of the block computed with the limits of the columns of the LaTeX array (and not with the contents of those columns), one may use the key `L`, `R` and `C` of the command `\Block`.

5 The rules

The usual techniques for the rules may be used in the environments of `nicematrix` (excepted `\vline`). However, there is some small differences with the classical environments.

¹¹One may consider that the default value of the first mandatory argument of `\Block` is `1-1`.

5.1 Some differences with the classical environments

5.1.1 The vertical rules

In the environments of `nicematrix`, the vertical rules specified by `|` in the preambles of the environments are never broken, even by an incomplete row or by a double horizontal rule specified by `\hline\hline` (there is no need to use `hhline`).

```
\begin{NiceTabular}{|c|c|} \hline
First & Second \\ \hline\hline
Peter & \\ \hline
Mary & George \\ \hline
\end{NiceTabular}
```

First	Second
Peter	
Mary	George

However, the vertical rules are not drawn in the blocks (created by `\Block`: cf. p. 4) nor in the corners (created by the key `corner`: cf. p. 10).

If you use `booktabs` (which provides `\toprule`, `\midrule`, `\bottomrule`, etc.) and if you really want to add vertical rules (which is not in the spirit of `booktabs`), you should notice that the vertical rules drawn by `nicematrix` are compatible with `booktabs`.

```
$\begin{NiceArray}{|cccc|} \toprule
a & b & c & d \\ \midrule
1 & 2 & 3 & 4 \\
1 & 2 & 3 & 4 \\ \bottomrule
\end{NiceArray}$
```

<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>
1	2	3	4
1	2	3	4

However, it's still possible to define a specifier (named, for instance, `I`) to draw vertical rules with the standard behaviour of `array`.

```
\newcolumnntype{I}{!{\vrule}}
```

However, in this case, it is probably more clever to add a command `\OnlyMainNiceMatrix` (cf. p. 45):

```
\newcolumnntype{I}{!{\OnlyMainNiceMatrix{\vrule}}}
```

5.1.2 The command `\cline`

The horizontal and vertical rules drawn by `\hline` and the specifier “`|`” make the array larger or wider by a quantity equal to the width of the rule (with `array` and also with `nicematrix`).

For historical reasons, this is not the case with the command `\cline`, as shown by the following example.

```
\setlength{\arrayrulewidth}{2pt}
\begin{tabular}{cccc} \hline
A&B&C&D \\ \cline{2-2}
A&B&C&D \\ \hline
\end{tabular}
```

A	B	C	D
A	<u>B</u>	C	D

In the environments of `nicematrix`, this situation is corrected (it's still possible to go to the standard behaviour of `\cline` with the key `standard-cline`).

```
\setlength{\arrayrulewidth}{2pt}
\begin{NiceTabular}{cccc} \hline
A&B&C&D \\ \cline{2}
A&B&C&D \\ \hline
\end{NiceTabular}
```

A	B	C	D
A	<u>B</u>	C	D

In the environments of `nicematrix`, an instruction `\cline{i}` is equivalent to `\cline{i-i}`.

5.2 The thickness and the color of the rules

The environments of `nicematrix` provide a key `rules/width` to set the width (in fact the thickness) of the rules in the current environment. In fact, this key merely sets the value of the length `\arrayrulewidth`.

It's well known that `colortbl` provides the command `\arrayrulecolor` in order to specify the color of the rules.

With `nicematrix`, it's possible to specify the color of the rules even when `colortbl` is not loaded. For sake of compatibility, the command is also named `\arrayrulecolor`. The environments of `nicematrix` also provide a key `rules/color` to fix the color of the rules in the current environment. This key sets the value locally (whereas `\arrayrulecolor` acts globally).

```
\begin{NiceTabular}{|ccc|}[rules/color=[gray]{0.9},rules/width=1pt]
\hline
rose & tulipe & lys \\
arum & iris & violette \\
muguet & dahlia & souci \\
\hline
\end{NiceTabular}
```

rose	tulipe	lys
arum	iris	violette
muguet	dahlia	souci

If one wishes to define new specifiers for columns in order to draw vertical rules (for example with a specific color or thicker than the standard rules), he should consider the command `\OnlyMainNiceMatrix` described on page 45.

5.3 The tools of `nicematrix` for the rules

Here are the tools provided by `nicematrix` for the rules.

- the keys `hlines`, `vlines`, `hvlines` and `hvlines-except-borders`;
- the specifier “|” in the preamble (for the environments with preamble);
- the command `\Hline`.

All these tools don't draw the rules in the blocks nor in the empty corners (when the key corners is used).

- These blocks are:
 - the blocks created by the command `\Block`¹² presented p. 4;
 - the blocks implicitly delimited by the continuous dotted lines created by `\Cdots`, `\Vdots`, etc. (cf. p. 23).
- The corners are created by the key `corners` explained below (see p. 10).

In particular, this remark explains the difference between the standard command `\hline` and the command `\Hline` provided by `nicematrix`.

5.3.1 The keys `hlines` and `vlines`

The keys `hlines` and `vlines` (which draw, of course, horizontal and vertical rules) take in as value a list of numbers which are the numbers of the rules to draw.¹³

In fact, for the environments with delimiters (such as `{pNiceMatrix}` or `{bNiceArray}`), the key `vlines` don't draw the exterior rules (this is certainly the expected behaviour).

```
$\begin{pNiceMatrix}[vlines,rules/width=0.2pt]
1 & 2 & 3 & 4 & 5 & 6 \\
1 & 2 & 3 & 4 & 5 & 6 \\
1 & 2 & 3 & 4 & 5 & 6 \\
\end{pNiceMatrix}$
```

1	2	3	4	5	6
1	2	3	4	5	6
1	2	3	4	5	6

¹²And also the command `\multicolumn` but it's recommended to use instead `\Block` in the environments of `nicematrix`.

¹³It's possible to put in that list some intervals of integers with the syntax `i-j`.

5.3.2 The keys `hvlines` and `hvlines-except-borders`

The key `hvlines` (no value) is the conjunction of the keys `hlines` and `vlines`.

```
\setlength{\arrayrulewidth}{1pt}
\begin{NiceTabular}{cccc}[hvlines,rules/color=blue]
rose      & tulipe & marguerite & dahlia \\
violette  & \Block[draw=red]{2-2}{\LARGE fleurs} & & souci \\
pervenche & & & lys \\
arum      & iris   & jacinthe  & muguet
\end{NiceTabular}
```

rose	tulipe	marguerite	dahlia
violette	fleurs		souci
pervenche			lys
arum	iris	jacinthe	muguet

The key `hvlines-except-borders` is similar to the key `hvlines` but does not draw the rules on the horizontal and vertical borders of the array.

5.3.3 The (empty) corners

The four **corners** of an array will be designed by NW, SW, NE and SE (*north west*, *south west*, *north east* and *south east*).

For each of these corners, we will call *empty corner* (or simply *corner*) the reunion of all the empty rectangles starting from the cell actually in the corner of the array.¹⁴

However, it's possible, for a cell without content, to require `nicemarix` to consider that cell as not empty with the key `\NotEmpty`.

In the example on the right (where B is in the center of a block of size 2×2), we have colored in blue the four (empty) corners of the array.

				A	
				A	A
			A		
		A	A	A	A
A	A	A	A	A	A
A	A	A	A	A	A
	A	A	A		
			A		
		B	A		

When the key `corners` is used, `nicematrix` computes the (empty) corners and these corners will be taken into account by the tools for drawing the rules (the rules won't be drawn in the corners).
Remark: In the previous versions of `nicematrix`, there was only a key `hvlines-except-corners` (now considered as obsolete).

¹⁴For sake of completeness, we should also say that a cell contained in a block (even an empty cell) is not taken into account for the determination of the corners. That behaviour is natural. The precise definition of a “non-empty cell” is given below (cf. p. 46).

```

\NiceMatrixOptions{cell-space-top-limit=3pt}
\begin{NiceTabular}{*{6}{c}}[corners,hvlines]
& & & & A & \\
& & & A & A & \\
& & & & & \\
& & & A & & \\
& & A & A & A & A & \\
A & A & A & A & A & A & \\
A & A & A & A & A & A & \\
& A & A & A & A & \\
& \Block{2-2}{B} & & A & \\
& & & A & \\
\end{NiceTabular}

```

					A
		A	A	A	
			A		
		A	A	A	A
A	A	A	A	A	A
A	A	A	A	A	A
	A	A	A		
	B		A		
			A		

It's also possible to provide to the key **corners** a (comma-separated) list of corners (designed by NW, SW, NE and SE).

```

\NiceMatrixOptions{cell-space-top-limit=3pt}
\begin{NiceTabular}{*{6}{c}}[corners=NE,hvlines]
1\\
1&1\\
1&2&1\\
1&3&3&1\\
1&4&6&4&1\\
& & & & & 1
\end{NiceTabular}

```

1					
1	1				
1	2	1			
1	3	3	1		
1	4	6	4	1	
					1

▷ The corners are also taken into account by the tools provided by **nicematrix** to color cells, rows and columns. These tools don't color the cells which are in the corners (cf. p. 14).

5.4 The command `\diagbox`

The command `\diagbox` (inspired by the package **diagbox**), allows, when it is used in a cell, to slash that cell diagonally downwards.¹⁵

```

$\begin{NiceArray}{*{5}{c}}[hvlines]
\diagbox{x}{y} & e & a & b & c \\
e & e & a & b & c \\
a & a & e & c & b \\
b & b & c & e & a \\
c & c & b & a & e
\end{NiceArray}$

```

$\begin{smallmatrix} y \\ x \end{smallmatrix}$	e	a	b	c
e	e	a	b	c
a	a	e	c	b
b	b	c	e	a
c	c	b	a	e

It's possible to use the command `\diagbox` in a `\Block`.

5.5 Dotted rules

In the environments of the package **nicematrix**, it's possible to use the command `\hdottedline` (provided by **nicematrix**) which is a counterpart of the classical commands `\hline` and `\hdashline` (the latter is a command of **arydshln**).

```

\begin{pNiceMatrix}
1 & 2 & 3 & 4 & 5 \\
\hdottedline
6 & 7 & 8 & 9 & 10 \\
11 & 12 & 13 & 14 & 15
\end{pNiceMatrix}

```

$$\left(\begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \\ \hdottedline 6 & 7 & 8 & 9 & 10 \\ 11 & 12 & 13 & 14 & 15 \end{array} \right)$$

In the environments with an explicit preamble (like `{NiceTabular}`, `{NiceArray}`, etc.), it's possible to draw a vertical dotted line with the specifier “:”.

¹⁵The author of this document considers that type of construction as graphically poor.

```

\left(\begin{NiceArray}{cccc:c}
1 & 2 & 3 & 4 & 5 \\
6 & 7 & 8 & 9 & 10 \\
11 & 12 & 13 & 14 & 15
\end{NiceArray}\right)

```

$$\left(\begin{array}{cccc:c} 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 10 \\ 11 & 12 & 13 & 14 & 15 \end{array} \right)$$

It's possible to change in `nicematrix` the letter used to specify a vertical dotted line with the option `letter-for-dotted-lines` available in `\NiceMatrixOptions`. Thus released, the letter “:” can be used otherwise (for example by the package `arydshln`¹⁶).

Remark: In the package `array` (on which the package `nicematrix` relies), horizontal and vertical rules make the array larger or wider by a quantity equal to the width of the rule¹⁷. In `nicematrix`, the dotted lines drawn by `\hdottedline` and “:” do likewise.

5.6 Commands for customized rules

New 6.5 It's possible to define commands and letters for customized rules with the key `custom-line` available in `\NiceMatrixOptions` and in the options of individual environments. That key takes in as argument a list of `key=value` pairs. First, there is two keys to define the tools which will be used to use that new type of rule.

- the key `command` is the name (without the backslashes) of a command that will be created by `nicematrix` and that will be available for the final user in order to draw horizontal rules (similarly to `\hline`);
- the key `letter` takes in as argument a letter¹⁸ that the user will use in the preamble of an environment with preamble (such as `\NiceTabular`) in order to specify a vertical rule.

For the description of the rule itself, there is three possibilities.

- *First possibility*

It's possible to specify composite rules, with a color and a color for the inter-rule space (as possible with `colortbl` for instance).

- the key `multiplicity` is the number to consecutive rules that will be drawn: for instance, a value of 2 will create double rules such those created by `\hline\hline` or `||` in the preamble of an environment;
- the key `color` sets the color of the rule ;
- the key `sep-color` sets the color between two successive rules (should be used only in conjunction with `multiplicity`).

- *Second possibility*

The key `dotted` forces a style with dotted rules such as those created by `\hdottedline` or the letter “:” in the preamble (cf. p. 11). The key `color` may be used also in that case.

- **New 6.6** *Third possibility*

It's possible to use the key `tikz` (if `Tikz` is loaded). In that case, the rule is drawn directly with `Tikz` by using as parameters the value of the key `tikz` which must be a list of `key=value` pairs which may be applied to a `Tikz` path.

By default, no space is reserved for the rule that will be drawn with `Tikz`. It possible to specify a reservation (horizontal for a vertical rule and vertical for an horizontal one) with the key `width`. That value of that key, is, in some ways, the width of the rule that will be drawn (`nicematrix` does not compute that width from the characteristics of the rule specified in `tikz`).

¹⁶However, one should remark that the package `arydshln` is not fully compatible with `nicematrix`.

¹⁷In fact, with `array`, this is true only for `\hline` and “|” but not for `\cline`: cf p. 8

¹⁸The following letters are forbidden: `lcrpmbVX:|()[]!@<>`

That system may be used, in particular, for the definition of commands and letters to draw rules with a specific color (and those rules will respect the blocks as do all rules of `nicematrix`).

```
\begin{NiceTabular}{lcIcIc}[custom-line = {letter=I, color=blue}]
\hline
      & \Block{1-3}{dimensions} & \\
      & L & l & h & \\
\hline
Product A & 3 & 1 & 2 & \\
Product B & 1 & 3 & 4 & \\
Product C & 5 & 4 & 1 & \\
\hline
\end{NiceTabular}
```

	dimensions		
	L	l	H
Product A	3	1	2
Product B	1	3	4
Product C	5	4	1

Here is an example of the key `tikz`.

```
\documentclass{article}
\usepackage{nicematrix,tikz}
\usetikzlibrary{decorations.pathmorphing}

\NiceMatrixOptions
{
  custom-line =
  {
    letter = I ,
    tikz = { decorate, decoration = { coil, aspect = 0 } } ,
    width = 2 mm
  }
}

\begin{document}
\begin{NiceTabular}{cIcIc}
one & two & three & \\
four & five & six & \\
seven & eight & nine & 
\end{NiceTabular}
\end{document}
```

one	⋈	two	⋈	three
four	⋈	five	⋈	six
seven	⋈	eight	⋈	nine

6 The color of the rows and columns

6.1 Use of `colortbl`

We recall that the package `colortbl` can be loaded directly with `\usepackage{colortbl}` or by loading `xcolor` with the key `table`: `\usepackage[table]{xcolor}`.

Since the package `nicematrix` is based on `array`, it's possible to use `colortbl` with `nicematrix`.

However, there is two drawbacks:

- The package `colortbl` patches `array`, leading to some incompatibilities (for instance with the command `\hdotsfor`).
 - The package `colortbl` constructs the array row by row, alternating colored rectangles, rules and contents of the cells. The resulting PDF is difficult to interpret by some PDF viewers and may lead to artefacts on the screen.
 - Some rules seem to disappear. This is because many PDF viewers give priority to graphical element drawn posteriorly (which is in the spirit of the “painting model” of PostScript and PDF). Concerning this problem, MuPDF (which is used, for instance, by SumatraPDF) gives better results than Adobe Reader).
 - A thin white line may appear between two cells of the same color. This phenomenon occurs when each cell is colored with its own instruction `fill` (the PostScript operator `fill` noted `f` in PDF). This is the case with `colortbl`: each cell is colored on its own, even when `\columncolor` or `\rowcolor` is used.
- As for this phenomenon, Adobe Reader gives better results than MuPDF.

The package `nicematrix` provides tools to avoid those problems.

6.2 The tools of `nicematrix` in the `\CodeBefore`

The package `nicematrix` provides some tools (independent of `colortbl`) to draw the colored panels first, and, then, the content of the cells and the rules. This strategy is more conform to the “painting model” of the formats PostScript and PDF and is more suitable for the PDF viewers. However, it requires several compilations.¹⁹

The extension `nicematrix` provides a key `code-before` for some code that will be executed before the drawing of the tabular.

An alternative syntax is provided: it’s possible to put the content of that `code-before` between the keywords `\CodeBefore` and `\Body` at the beginning of the environment.

```
\begin{pNiceArray}{preamble}
\CodeBefore
  instructions of the code-before
\Body
  contents of the environment
\end{pNiceArray}
```

New commands are available in that `\CodeBefore`: `\cellcolor`, `\rectanglecolor`, `\rowcolor`, `\columncolor`, `\rowcolors`, `\rowlistcolors`, `\chessboardcolors` and `arraycolor`.²⁰

All these commands accept an optional argument (between square brackets and in first position) which is the color model for the specification of the colors.

These commands don’t color the cells which are in the “corners” if the key `corners` is used. This key has been described p. 10.

- The command `\cellcolor` takes its name from the command `\cellcolor` of `colortbl`.
This command takes in as mandatory arguments a color and a list of cells, each of which with the format i - j where i is the number of the row and j the number of the column of the cell.

¹⁹If you use Overleaf, Overleaf will do automatically the right number of compilations.

²⁰Remark that, in the `\CodeBefore`, PGF/Tikz nodes of the form “(i-|j)” are also available to indicate the position to the potential rules: cf. p. 42.

```

\begin{NiceTabular}{|c|c|c|}
\CodeBefore
  \cellcolor[HTML]{FFFF88}{3-1,2-2,1-3}
\Body
\hline
a & b & c \\ \hline
e & f & g \\ \hline
h & i & j \\ \hline
\end{NiceTabular}

```

a	b	c
e	f	g
h	i	j

- The command `\rectanglecolor` takes three mandatory arguments. The first is the color. The second is the upper-left cell of the rectangle and the third is the lower-right cell of the rectangle.

```

\begin{NiceTabular}{|c|c|c|}
\CodeBefore
  \rectanglecolor{blue!15}{2-2}{3-3}
\Body
\hline
a & b & c \\ \hline
e & f & g \\ \hline
h & i & j \\ \hline
\end{NiceTabular}

```

a	b	c
e	f	g
h	i	j

- The command `\arraycolor` takes in as mandatory argument a color and color the whole tabular with that color (excepted the potential exterior rows and columns: cf. p. 21). It's only a particular case of `\rectanglecolor`.
- The command `\chessboardcolors` takes in as mandatory arguments two colors and it colors the cells of the tabular in quincunx with these colors.

```

$\begin{pNiceMatrix}[r,margin]
\CodeBefore
  \chessboardcolors{red!15}{blue!15}
\Body
1 & -1 & 1 \\
-1 & 1 & -1 \\
1 & -1 & 1
\end{pNiceMatrix}$

```

1	-1	1
-1	1	-1
1	-1	1

We have used the key `r` which aligns all the columns rightwards (cf. p. 36).

- The command `\rowcolor` takes its name from the command `\rowcolor` of `colortbl`. Its first mandatory argument is the color and the second is a comma-separated list of rows or interval of rows with the form `a-b` (an interval of the form `a-` represent all the rows from the row `a` until the end).

```

$\begin{NiceArray}{lll}[hvlines]
\CodeBefore
  \rowcolor{red!15}{1,3-5,8-}
\Body
a_1 & b_1 & c_1 \\
a_2 & b_2 & c_2 \\
a_3 & b_3 & c_3 \\
a_4 & b_4 & c_4 \\
a_5 & b_5 & c_5 \\
a_6 & b_6 & c_6 \\
a_7 & b_7 & c_7 \\
a_8 & b_8 & c_8 \\
a_9 & b_9 & c_9 \\
a_{10} & b_{10} & c_{10} \\
\end{NiceArray}$

```

a_1	b_1	c_1
a_2	b_2	c_2
a_3	b_3	c_3
a_4	b_4	c_4
a_5	b_5	c_5
a_6	b_6	c_6
a_7	b_7	c_7
a_8	b_8	c_8
a_9	b_9	c_9
a_{10}	b_{10}	c_{10}

- The command `\columncolor` takes its name from the command `\columncolor` of `colortbl`. Its syntax is similar to the syntax of `\rowcolor`.
- The command `\rowcolors` (with a *s*) takes its name from the command `\rowcolors` of `xcolor`²¹. The *s* emphasizes the fact that there is *two* colors. This command colors alternately the rows of the tabular with the tow colors (provided in second and third argument), beginning with the row whose number is given in first (mandatory) argument.

In fact, the first (mandatory) argument is, more generally, a comma separated list of intervals describing the rows involved in the action of `\rowcolors` (an interval of the form *i-* describes in fact the interval of all the rows of the tabular, beginning with the row *i*).

The last argument of `\rowcolors` is an optional list of pairs *key=value* (the optional argument in the first position corresponds to the colorimetric space). The available keys are `cols`, `restart` and `respect-blocks`.

- The key `cols` describes a set of columns. The command `\rowcolors` will color only the cells of these columns. The value is a comma-separated list of intervals of the form *i-j* (where *i* or *j* may be replaced by *).
- With the key `restart`, each interval of rows (specified by the first mandatory argument) begins with the same color.²²
- With the key `respect-blocks` the “rows” alternately colored may extend over several rows if they have to incorporate blocks (created with the command `\Block`: cf. p. 4).

```
\begin{NiceTabular}{clr}[hvlines]
\CodeBefore
  \rowcolors[gray]{2}{0.8}{}[cols=2-3,restart]
\Body
\Block{1-*}{Results} \
John & 12 \
Stephen & 8 \
Sarah & 18 \
Ashley & 20 \
Henry & 14 \
Madison & 15
\end{NiceTabular}
```

Results		
A	John	12
	Stephen	8
B	Sarah	18
	Ashley	20
	Henry	14
	Madison	15

```
\begin{NiceTabular}{lr}[hvlines]
\CodeBefore
  \rowcolors{1}{blue!10}{}[respect-blocks]
\Body
\Block{2-1}{John} & 12 \
& 13 \
Steph & 8 \
\Block{3-1}{Sarah} & 18 \
& 17 \
& 15 \
Ashley & 20 \
Henry & 14 \
\Block{2-1}{Madison} & 15 \
& 19
\end{NiceTabular}
```

John	12
	13
Steph	8
Sarah	18
	17
	15
Ashley	20
Henry	14
Madison	15
	19

²¹The command `\rowcolors` of `xcolor` is available when `xcolor` is loaded with the option `table`. That option also loads the package `colortbl`.

²²Otherwise, the color of a given row relies only upon the parity of its absolute number.

- The extension `nicematrix` provides also a command `\rowlistcolors`. This command generalises the command `\rowcolors`: instead of two successive arguments for the colors, this command takes in an argument which is a (comma-separated) list of colors. In that list, the symbol `=` represent a color identical to the previous one.

```
\begin{NiceTabular}{c}
\CodeBefore
  \rowlistcolors{1}{red!15,blue!15,green!15}
\Body
Peter \\
James \\
Abigail \\
Elisabeth \\
Claudius \\
Jane \\
Alexandra \\
\end{NiceTabular}
```

Peter
James
Abigail
Elisabeth
Claudius
Jane
Alexandra

We recall that all the color commands we have described don't color the cells which are in the "corners". In the following example, we use the key `corners=NE` to require the determination of the corner *north east* (NE).

```
\begin{NiceTabular}{cccccc}[corners=NE,margin,hvlines,first-row,first-col]
\CodeBefore
  \rowlistcolors{1}{blue!15, }
\Body
& 0 & 1 & 2 & 3 & 4 & 5 & 6 \\
0 & 1 & \\
1 & 1 & 1 & \\
2 & 1 & 2 & 1 & \\
3 & 1 & 3 & 3 & 1 & \\
4 & 1 & 4 & 6 & 4 & 1 & \\
5 & 1 & 5 & 10 & 10 & 5 & 1 & \\
6 & 1 & 6 & 15 & 20 & 15 & 6 & 1 \\
\end{NiceTabular}
```

	0	1	2	3	4	5	6
0	1						
1	1	1					
2	1	2	1				
3	1	3	3	1			
4	1	4	6	4	1		
5	1	5	10	10	5	1	
6	1	6	15	20	15	6	1

One should remark that all the previous commands are compatible with the commands of `booktabs` (`\toprule`, `\midrule`, `\bottomrule`, etc). However, `booktabs` is not loaded by `nicematrix`.

```
\begin{NiceTabular}[c]{lSSSS}
\CodeBefore
  \rowcolor{red!15}{1-2}
  \rowcolors{3}{blue!15}{-}
\Body
\toprule
\Block{2-1}{Product} &
\Block{1-3}{dimensions (cm)} & & &
\Block{2-1}{\rotate Price} \\
\cmidrule{rl}{2-4}
& L & l & h & \\
\midrule
small & 3 & 5.5 & 1 & 30 \\
standard & 5.5 & 8 & 1.5 & 50.5 \\
premium & 8.5 & 10.5 & 2 & 80 \\
extra & 8.5 & 10 & 1.5 & 85.5 \\
special & 12 & 12 & 0.5 & 70 \\
\bottomrule
\end{NiceTabular}
```

Product	dimensions (cm)			Price
	L	l	h	
small	3	5.5	1	30
standard	5.5	8	1.5	50.5
premium	8.5	10.5	2	80
extra	8.5	10	1.5	85.5
special	12	12	0.5	70

We have used the type of column `S` of `siunitx`.

6.3 Color tools with the syntax of colortbl

It's possible to access the preceding tools with a syntax close to the syntax of `colortbl`. For that, one must use the key `colortbl-like` in the current environment.²³ There are three commands available (they are inspired by `colortbl` but are *independent* of `colortbl`):

- `\cellcolor` which colorizes a cell;²⁴
- `\rowcolor` which must be used in a cell and which colorizes the end of the row;
- `\columncolor` which must be used in the preamble of the environment with the same syntax as the corresponding command of `colortbl` (however, unlike the command `\columncolor` of `colortbl`, this command `\columncolor` can appear within another command, itself used in the preamble of the array).

```
\NewDocumentCommand { \Blue } { } { \columncolor{blue!15} }
\begin{NiceTabular}[colortbl-like]{>{\Blue}c>{\Blue}cc}
\toprule
\rowcolor{red!15}
Last name & First name & Birth day \\
\midrule
Achard & Jacques & 5 juin 1962 \\
Lefebvre & Mathilde & 23 mai 1988 \\
Vanesse & Stephany & 30 octobre 1994 \\
Dupont & Chantal & 15 janvier 1998 \\
\bottomrule
\end{NiceTabular}
```

Last name	First name	Birth day
Achard	Jacques	5 juin 1962
Lefebvre	Mathilde	23 mai 1988
Vanesse	Stephany	30 octobre 1994
Dupont	Chantal	15 janvier 1998

7 The command `\RowStyle`

The command `\RowStyle` takes in as argument some formatting intructions that will be applied to each cell on the rest of the current row.

That command also takes in as optional argument (between square brackets) a list of *key=value* pairs.

- **New 6.3** The key `nb-rows` sets the number of rows to which the specifications of the current command will apply.
- The keys `cell-space-top-limit`, `cell-space-bottom-limit` and `cell-space-limits` are available with the same meaning that the corresponding global keys (cf. p. 2).
- **New 6.3** The key `rowcolor` sets the color of the background and the key `color` sets the color of the text.²⁵

²³Up to now, this key is *not* available in `\NiceMatrixOptions`.

²⁴However, this command `\cellcolor` will delete the following spaces, which does not the command `\cellcolor` of `colortbl`.

²⁵The key `color` uses the command `\color` but inserts also an instruction `\leavevmode` before. This instruction prevents a extra vertical space in the cells which belong to columns of type `p`, `b`, `m` and `X` (which start in vertical mode).

- **New 6.3** The key `bold` enforces bold characters for the cells of the row, both in math mode and text mode.

```
\begin{NiceTabular}{cccc}
\hline
\RowStyle[cell-space-limits=3pt]{\rotate}
first & second & third & fourth \\
\RowStyle[nb-rows=2,rowcolor=blue!50,color=white]{\sffamily}
1 & 2 & 3 & 4 \\
I & II & III & IV
\end{NiceTabular}
```

first	second	third	fourth
1	2	3	4
I	II	III	IV

The command `\rotate` is described p. 36.

8 The width of the columns

8.1 Basic tools

In the environments with an explicit preamble (like `{NiceTabular}`, `{NiceArray}`, etc.), it's possible to fix the width of a given column with the standard letters `w`, `W`, `p`, `b` and `m` of the package `array`.

```
\begin{NiceTabular}{Wc{2cm}cc}[hvlines]
Paris & New York & Madrid \\
Berlin & London & Roma \\
Rio & Tokyo & Oslo
\end{NiceTabular}
```

Paris	New York	Madrid
Berlin	London	Roma
Rio	Tokyo	Oslo

In the environments of `nicematrix`, it's also possible to fix the *minimal* width of all the columns (excepted the potential exterior columns: cf. p. 21) directly with the key `columns-width`.

```
$\begin{pNiceMatrix}[columns-width = 1cm]
1 & 12 & -123 \\
12 & 0 & 0 \\
4 & 1 & 2
\end{pNiceMatrix}$
```

$$\begin{pmatrix} 1 & 12 & -123 \\ 12 & 0 & 0 \\ 4 & 1 & 2 \end{pmatrix}$$

Note that the space inserted between two columns (equal to `2 \tabcolsep` in `{NiceTabular}` and to `2 \arraycolsep` in the other environments) is not suppressed (of course, it's possible to suppress this space by setting `\tabcolsep` or `\arraycolsep` equal to 0 pt before the environment).

It's possible to give the special value `auto` to the option `columns-width`: all the columns of the array will have a width equal to the widest cell of the array.²⁶

```
$\begin{pNiceMatrix}[columns-width = auto]
1 & 12 & -123 \\
12 & 0 & 0 \\
4 & 1 & 2
\end{pNiceMatrix}$
```

$$\begin{pmatrix} 1 & 12 & -123 \\ 12 & 0 & 0 \\ 4 & 1 & 2 \end{pmatrix}$$

Without surprise, it's possible to fix the minimal width of the columns of all the arrays of a current scope with the command `\NiceMatrixOptions`.

```
\NiceMatrixOptions{columns-width=10mm}
$\begin{pNiceMatrix}
a & b \\ c & d
\end{pNiceMatrix}
=
\begin{pNiceMatrix}
1 & 1245 \\ 345 & 2
\end{pNiceMatrix}$
```

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} = \begin{pmatrix} 1 & 1245 \\ 345 & 2 \end{pmatrix}$$

²⁶The result is achieved with only one compilation (but PGF/Tikz will have written informations in the `aux` file and a message requiring a second compilation will appear).

But it's also possible to fix a zone where all the matrices will have their columns of the same width, equal to the widest cell of all the matrices. This construction uses the environment `{NiceMatrixBlock}` with the option `auto-columns-width`²⁷. The environment `{NiceMatrixBlock}` has no direct link with the command `\Block` presented previously in this document (cf. p. 4).

```
\begin{NiceMatrixBlock}[auto-columns-width]
$\begin{array}{c}
\begin{bNiceMatrix}
9 & 17 \\ -2 & 5
\end{bNiceMatrix} \\
\begin{bNiceMatrix}
1 & 1245345 \\ 345 & 2
\end{bNiceMatrix}
\end{array}$
\end{NiceMatrixBlock}
```

$$\begin{bmatrix} 9 & 17 \\ -2 & 5 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 1245345 \\ 345 & 2 \end{bmatrix}$$

8.2 The columns V of varwidth

New 6.3

Let's recall first the behaviour of the environment `{varwidth}` of the eponymous package `varwidth`. That environment is similar to the classical environment `{minipage}` but the width provided in the argument is only the *maximal* width of the created box. In the general case, the width of the box constructed by an environment `{varwidth}` is the natural width of its contents.

That point is illustrated on the following examples.

```
\fbox{%
\begin{varwidth}{8cm}
\begin{itemize}
\item first item
\item second item
\end{itemize}
\end{varwidth}}
```

- first item
- second item

```
\fbox{%
\begin{minipage}{8cm}
\begin{itemize}
\item first item
\item second item
\end{itemize}
\end{minipage}}
```

- first item
- second item

The package `varwidth` provides also the column type `V`. A column of type `V{<dim>}` encapsulates all its cells in a `{varwidth}` with the argument `<dim>` (and does also some tuning).

When the package `varwidth` is loaded, the columns `V` of `varwidth` are supported by `nicematrix`. Concerning `nicematrix`, one of the interests of this type of columns is that, for a cell of a column of type `V`, the PGF/Tikz node created by `nicematrix` for the content of that cell has a width adjusted to the content of the cell : cf. p. 40. If the content of the cell is empty, the cell will be considered as empty by `nicematrix` in the construction of the dotted lines and the «empty corners» (that's not the case with a cell of a column `p`, `m` or `b`).

```
\begin{NiceTabular}[corners=NW,hvlines]{V{3cm}V{3cm}V{3cm}}
& some very very very long text & some very very very long text \\
some very very very long text & \\
some very very very long text & \\
\end{NiceTabular}
```

²⁷At this time, this is the only usage of the environment `{NiceMatrixBlock}` but it may have other usages in the future.

	some very very very long text	some very very very long text
some very very very long text		
some very very very long text		

One should remark that the extension `varwidth` (at least in its version 0.92) has some problems: for instance, with LuaLaTeX, it does not work when the content begins with `\color`.

8.3 The columns X

The environment `{NiceTabular}` provides **X** columns similar to those provided by the environment `{tabularx}` of the eponymous package.

The required width of the tabular may be specified with the key `width` (in `{NiceTabular}` or in `\NiceMatrixOptions`). The initial value of this parameter is `\linewidth` (and not `\textwidth`).

For sake of similarity with the environment `{tabularx}`, `nicematrix` also provides an environment `{NiceTabularX}` with a first mandatory argument which is the width of the tabular.²⁸

As with the packages `tabu` and `tabularray`, the specifier **X** takes in an optional argument (between square brackets) which is a list of keys.

- It's possible to give a weight for the column by providing a positive integer directly as argument of the specifier **X**. For example, a column `X[2]` will have a width double of the width of a column **X** (which has a weight equal to 1).²⁹
- It's possible to specify an horizontal alignment with one of the letters `l`, `c` and `r` (which insert respectively `\raggedright`, `\centering` and `\raggedleft` followed by `\arraybackslash`).
- It's possible to specify a vertical alignment with one of the keys `t` (alias `p`), `m` and `b` (which construct respectively columns of type `p`, `m` and `b`). The default value is `t`.

```
\begin{NiceTabular}[width=9cm]{X[2,l]X[1]}[hvlines]
a rather long text which fits on several lines
& a rather long text which fits on several lines \\
a shorter text & a shorter text
\end{NiceTabular}
```

a rather long text which fits on several lines	a rather long text which fits on several lines
a shorter text	a shorter text

9 The exterior rows and columns

The options `first-row`, `last-row`, `first-col` and `last-col` allow the composition of exterior rows and columns in the environments of `nicematrix`. It's particularly interesting for the (mathematical) matrices.

A potential “first row” (exterior) has the number 0 (and not 1). Idem for the potential “first column”.

²⁸If `tabularx` is loaded, one must use `{NiceTabularX}` (and not `{NiceTabular}`) in order to use the columns **X** (this point comes from a conflict in the definitions of the specifier **X**).

²⁹The negative values of the weight, as provided by `tabu` (which is now obsolete), are *not* supported by `nicematrix`. If such a value is used, an error will be raised.

```

 $\begin{pNiceMatrix}[first-row,last-row,first-col,last-col,nullify-dots]$ 
& C_1 & & \Cdots & & C_4 & & \\
L_1 & a_{11} & a_{12} & a_{13} & a_{14} & L_1 & & \\
\vdots & a_{21} & a_{22} & a_{23} & a_{24} & \vdots & & \\
& a_{31} & a_{32} & a_{33} & a_{34} & & & \\
L_4 & a_{41} & a_{42} & a_{43} & a_{44} & L_4 & & \\
& C_1 & & \Cdots & & C_4 & & \\
\end{pNiceMatrix}

```

$$\begin{array}{c}
C_1 \dots\dots\dots C_4 \\
L_1 \left(\begin{array}{cccc} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{array} \right) L_1 \\
\vdots \\
\vdots \\
L_4 \left(\begin{array}{cccc} a_{41} & a_{42} & a_{43} & a_{44} \end{array} \right) L_4 \\
C_1 \dots\dots\dots C_4
\end{array}$$

The dotted lines have been drawn with the tools presented p. 23.

We have several remarks to do.

- For the environments with an explicit preamble (i.e. `{NiceTabular}`, `{NiceArray}` and its variants), no letter must be given in that preamble for the potential first column and the potential last column: they will automatically (and necessarily) be of type `r` for the first column and `l` for the last one.³⁰
- One may wonder how `nicematrix` determines the number of rows and columns which are needed for the composition of the “last row” and “last column”.
 - For the environments with explicit preamble, like `{NiceTabular}` and `{pNiceArray}`, the number of columns can obviously be computed from the preamble.
 - When the option `light-syntax` (cf. p. 38) is used, `nicematrix` has, in any case, to load the whole body of the environment (and that’s why it’s not possible to put verbatim material in the array with the option `light-syntax`). The analysis of this whole body gives the number of rows (but not the number of columns).
 - In the other cases, `nicematrix` compute the number of rows and columns during the first compilation and write the result in the `aux` file for the next run.

However, it’s possible to provide the number of the last row and the number of the last column as values of the options `last-row` and `last-col`, tending to an acceleration of the whole compilation of the document. That’s what we will do throughout the rest of the document.

It’s possible to control the appearance of these rows and columns with options `code-for-first-row`, `code-for-last-row`, `code-for-first-col` and `code-for-last-col`. These options specify tokens that will be inserted before each cell of the corresponding row or column.

```

\NiceMatrixOptions{code-for-first-row = \color{red},
                  code-for-first-col = \color{blue},
                  code-for-last-row = \color{green},
                  code-for-last-col = \color{magenta}}
 $\begin{pNiceArray}[cc|cc][first-row,last-row=5,first-col,last-col,nullify-dots]$ 
& C_1 & & \Cdots & & C_4 & & \\
L_1 & a_{11} & a_{12} & a_{13} & a_{14} & L_1 & & \\
\vdots & a_{21} & a_{22} & a_{23} & a_{24} & \vdots & & \\
\hline

```

³⁰The users wishing exterior columns with another type of alignment should consider the command `\SubMatrix` available in the `\CodeAfter` (cf. p. 29).

```

& a_{31} & a_{32} & a_{33} & a_{34} & & \\
L_4 & a_{41} & a_{42} & a_{43} & a_{44} & L_4 & \\
& C_1 & \Cdots & & C_4 & & \\
\end{pNiceArray}$

```

$$\begin{array}{c}
\textcolor{red}{C_1} \cdots \cdots \cdots \textcolor{red}{C_4} \\
\textcolor{blue}{L_1} \left(\begin{array}{cc|cc} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{array} \right) \textcolor{magenta}{L_1} \\
\textcolor{blue}{L_4} \left(\begin{array}{cc|cc} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{array} \right) \textcolor{magenta}{L_4} \\
\textcolor{green}{C_1} \cdots \cdots \cdots \textcolor{green}{C_4}
\end{array}$$

Remarks

- As shown in the previous example, the horizontal and vertical rules don't extend in the exterior rows and columns.
However, if one wishes to define new specifiers for columns in order to draw vertical rules (for example thicker than the standard rules), he should consider the command `\OnlyMainNiceMatrix` described on page 45.
- A specification of color present in `code-for-first-row` also applies to a dotted line drawn in that exterior “first row” (excepted if a value has been given to `xdots/color`). Idem for the other exterior rows and columns.
- Logically, the potential option `columns-width` (described p. 19) doesn't apply to the “first column” and “last column”.
- For technical reasons, it's not possible to use the option of the command `\\` after the “first row” or before the “last row”. The placement of the delimiters would be wrong. If you are looking for a workaround, consider the command `\SubMatrix` in the `\CodeAfter` described p. 29.

10 The continuous dotted lines

Inside the environments of the package `nicematrix`, new commands are defined: `\Ldots`, `\Cdots`, `\Vdots`, `\Ddots`, and `\Iddots`. These commands are intended to be used in place of `\dots`, `\cdots`, `\vdots`, `\ddots` and `\iddots`.³¹

Each of them must be used alone in the cell of the array and it draws a dotted line between the first non-empty cells³² on both sides of the current cell. Of course, for `\Ldots` and `\Cdots`, it's an horizontal line; for `\Vdots`, it's a vertical line and for `\Ddots` and `\Iddots` diagonal ones. It's possible to change the color of these lines with the option `color`.³³

```

\begin{bNiceMatrix}
a_1 & & \Cdots & & & & a_1 & \\
\Vdots & a_2 & & \Cdots & & a_2 & & \\
& & \Vdots & & \Ddots[color=red] & & & \\
\\
a_1 & & a_2 & & & & & a_n
\end{bNiceMatrix}

```

$$\begin{bmatrix}
a_1 & \cdots & \cdots & \cdots & a_1 \\
\vdots & & & & \\
& a_2 & \cdots & \cdots & a_2 \\
& \vdots & & & \\
a_1 & a_2 & & & a_n
\end{bmatrix}$$

³¹The command `\iddots`, defined in `nicematrix`, is a variant of `\ddots` with dots going forward. If `mathdots` is loaded, the version of `mathdots` is used. It corresponds to the command `\adots` of `unicode-math`.

³²The precise definition of a “non-empty cell” is given below (cf. p. 46).

³³It's also possible to change the color of all these dotted lines with the option `xdots/color` (`xdots` to remind that it works for `\Cdots`, `\Ldots`, `\Vdots`, etc.): cf. p. 27.

In order to represent the null matrix, one can use the following code:

```
\begin{bNiceMatrix}
0      & \Cdots & 0      & \\
\Vdots &         & \Vdots & \\
0      & \Cdots & 0      & \\
\end{bNiceMatrix}
```

$$\begin{bmatrix} 0 & \cdots & 0 \\ \vdots & & \vdots \\ 0 & \cdots & 0 \end{bmatrix}$$

However, one may want a larger matrix. Usually, in such a case, the users of LaTeX add a new row and a new column. It's possible to use the same method with `nicematrix`:

```
\begin{bNiceMatrix}
0      & \Cdots & \Cdots & 0      & \\
\Vdots &         &         & \Vdots & \\
\Vdots &         &         & \Vdots & \\
0      & \Cdots & \Cdots & 0      & \\
\end{bNiceMatrix}
```

$$\begin{bmatrix} 0 & \cdots & \cdots & 0 \\ \vdots & & & \vdots \\ \vdots & & & \vdots \\ 0 & \cdots & \cdots & 0 \end{bmatrix}$$

In the first column of this example, there are two instructions `\Vdots` but, of course, only one dotted line is drawn.

In fact, in this example, it would be possible to draw the same matrix more easily with the following code:

```
\begin{bNiceMatrix}
0      & \Cdots &         & 0      & \\
\Vdots &         &         & \Vdots & \\
         &         &         & \Vdots & \\
0      &         & \Cdots & 0      & \\
\end{bNiceMatrix}
```

$$\begin{bmatrix} 0 & \cdots & & 0 \\ \vdots & & & \vdots \\ & & & \vdots \\ 0 & & \cdots & 0 \end{bmatrix}$$

There are also other means to change the size of the matrix. Someone might want to use the optional argument of the command `\` for the vertical dimension and a command `\hspace*` in a cell for the horizontal dimension.³⁴

However, a command `\hspace*` might interfere with the construction of the dotted lines. That's why the package `nicematrix` provides a command `\Hspace` which is a variant of `\hspace` transparent for the dotted lines of `nicematrix`.

```
\begin{bNiceMatrix}
0      & \Cdots & \Hspace*{1cm} & 0      & \\
\Vdots &         &         & \Vdots & \\
0      & \Cdots &         & 0      & \\
\end{bNiceMatrix}
```

$$\begin{bmatrix} 0 & \cdots & & 0 \\ \vdots & & & \vdots \\ 0 & \cdots & & 0 \end{bmatrix}$$

10.1 The option `nullify-dots`

Consider the following matrix composed classically with the environment `{pmatrix}` of `amsmath`.

```
$A = \begin{pmatrix}
h & i & j & k & l & m \\
x & & & & & x
\end{pmatrix}$
```

$$A = \begin{pmatrix} h & i & j & k & l & m \\ x & & & & & x \end{pmatrix}$$

³⁴In `nicematrix`, one should use `\hspace*` and not `\hspace` for such an usage because `nicematrix` loads `array`. One may also remark that it's possible to fix the width of a column by using the environment `{NiceArray}` (or one of its variants) with a column of type `w` or `W`: see p. 19

If we add `\ldots` instructions in the second row, the geometry of the matrix is modified.

```
$B = \begin{pmatrix}
h & i & j & k & l & m \\
x & \ldots & \ldots & \ldots & \ldots & x
\end{pmatrix}
```

$$B = \begin{pmatrix} h & i & j & k & l & m \\ x & \dots & \dots & \dots & \dots & x \end{pmatrix}$$

By default, with `nicematrix`, if we replace `{pmatrix}` by `{pNiceMatrix}` and `\ldots` by `\Ldots`, the geometry of the matrix is not changed.

```
$C = \begin{pNiceMatrix}
h & i & j & k & l & m \\
x & \Ldots & \Ldots & \Ldots & \Ldots & x
\end{pNiceMatrix}
```

$$C = \begin{pmatrix} h & i & j & k & l & m \\ x & \dots & \dots & \dots & \dots & x \end{pmatrix}$$

However, one may prefer the geometry of the first matrix A and would like to have such a geometry with a dotted line in the second row. It's possible by using the option `nullify-dots` (and only one instruction `\Ldots` is necessary).

```
$D = \begin{pNiceMatrix}[nullify-dots]
h & i & j & k & l & m \\
x & \Ldots & & & & x
\end{pNiceMatrix}
```

$$D = \begin{pmatrix} h & i & j & k & l & m \\ x & \dots & & & & x \end{pmatrix}$$

The option `nullify-dots` smashes the instructions `\Ldots` (and the variants) horizontally but also vertically.

10.2 The commands `\Hdotsfor` and `\Vdotsfor`

Some people commonly use the command `\hdotsfor` of `amsmath` in order to draw horizontal dotted lines in a matrix. In the environments of `nicematrix`, one should use instead `\Hdotsfor` in order to draw dotted lines similar to the other dotted lines drawn by the package `nicematrix`.

As with the other commands of `nicematrix` (like `\Cdots`, `\Ldots`, `\Vdots`, etc.), the dotted line drawn with `\Hdotsfor` extends until the contents of the cells on both sides.

```
$\begin{pNiceMatrix}
1 & 2 & 3 & 4 & 5 \\
1 & \Hdotsfor{3} & 5 \\
1 & 2 & 3 & 4 & 5 \\
1 & 2 & 3 & 4 & 5
\end{pNiceMatrix}
```

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 1 & \dots & \dots & \dots & 5 \\ 1 & 2 & 3 & 4 & 5 \\ 1 & 2 & 3 & 4 & 5 \end{pmatrix}$$

However, if these cells are empty, the dotted line extends only in the cells specified by the argument of `\Hdotsfor` (by design).

```
$\begin{pNiceMatrix}
1 & 2 & 3 & 4 & 5 \\
& \Hdotsfor{3} \\
1 & 2 & 3 & 4 & 5 \\
1 & 2 & 3 & 4 & 5
\end{pNiceMatrix}
```

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ & \dots & \dots & \dots & \\ 1 & 2 & 3 & 4 & 5 \\ 1 & 2 & 3 & 4 & 5 \end{pmatrix}$$

Remark: Unlike the command `\hdotsfor` of `amsmath`, the command `\Hdotsfor` may be used even when the package `colortbl`³⁵ is loaded (but you might have problem if you use `\rowcolor` on the same row as `\Hdotsfor`).

The package `nicematrix` also provides a command `\Vdotsfor` similar to `\Hdotsfor` but for the vertical dotted lines. The following example uses both `\Hdotsfor` and `\Vdotsfor`:

³⁵We recall that when `xcolor` is loaded with the option `table`, the package `colortbl` is loaded.

```

\begin{bNiceMatrix}
C[a_1,a_1] & \Cdots & C[a_1,a_n]
& \hspace*{20mm} & C[a_1,a_1^{(p)}] & \Cdots & C[a_1,a_n^{(p)}] \\
\vdots & \ddots & \vdots
& \Hdotsfor{1} & \vdots & \ddots & \vdots \\
C[a_n,a_1] & \Cdots & C[a_n,a_n]
& & C[a_n,a_1^{(p)}] & \Cdots & C[a_n,a_n^{(p)}] \\
\rule{0pt}{15mm}\NotEmpty & \Vdotsfor{1} & & \ddots & & \Vdotsfor{1} \\
C[a_1^{(p)},a_1] & \Cdots & C[a_1^{(p)},a_n]
& & C[a_1^{(p)},a_1^{(p)}] & \Cdots & C[a_1^{(p)},a_n^{(p)}] \\
\vdots & \ddots & \vdots
& \Hdotsfor{1} & \vdots & \ddots & \vdots \\
C[a_n^{(p)},a_1] & \Cdots & C[a_n^{(p)},a_n]
& & C[a_n^{(p)},a_1^{(p)}] & \Cdots & C[a_n^{(p)},a_n^{(p)}] \\
\end{bNiceMatrix}

```

$$\begin{bmatrix}
C[a_1, a_1] & \cdots & C[a_1, a_n] & & C[a_1, a_1^{(p)}] & \cdots & C[a_1, a_n^{(p)}] \\
\vdots & \ddots & \vdots & & \vdots & \ddots & \vdots \\
C[a_n, a_1] & \cdots & C[a_n, a_n] & \cdots & C[a_n, a_1^{(p)}] & \cdots & C[a_n, a_n^{(p)}] \\
& \vdots & & \ddots & \vdots & & \vdots \\
C[a_1^{(p)}, a_1] & \cdots & C[a_1^{(p)}, a_n] & & C[a_1^{(p)}, a_1^{(p)}] & \cdots & C[a_1^{(p)}, a_n^{(p)}] \\
\vdots & \ddots & \vdots & & \vdots & \ddots & \vdots \\
C[a_n^{(p)}, a_1] & \cdots & C[a_n^{(p)}, a_n] & \cdots & C[a_n^{(p)}, a_1^{(p)}] & \cdots & C[a_n^{(p)}, a_n^{(p)}]
\end{bmatrix}$$

10.3 How to generate the continuous dotted lines transparently

Imagine you have a document with a great number of mathematical matrices with ellipsis. You may wish to use the dotted lines of `nicematrix` without having to modify the code of each matrix. It's possible with the keys `renew-dots` and `renew-matrix`.³⁶

- The option `renew-dots`

With this option, the commands `\ldots`, `\cdots`, `\vdots`, `\ddots`, `\iddots`³¹ and `\hdotsfor` are redefined within the environments provided by `nicematrix` and behave like `\Ldots`, `\Cdots`, `\Vdots`, `\Ddots`, `\Iddots` and `\Hdotsfor`; the command `\dots` (“automatic dots” of `amsmath`) is also redefined to behave like `\Ldots`.

- The option `renew-matrix`

With this option, the environment `{matrix}` is redefined and behave like `{NiceMatrix}`, and so on for the five variants.

Therefore, with the keys `renew-dots` and `renew-matrix`, a classical code gives directly the output of `nicematrix`.

```

\NiceMatrixOptions{renew-dots,renew-matrix}
\begin{pmatrix}
1 & \cdots & \cdots & 1 \\
0 & \ddots & & \vdots \\
\vdots & \ddots & \ddots & \vdots \\
0 & \cdots & 0 & 1
\end{pmatrix}

```

$$\begin{pmatrix}
1 & \cdots & \cdots & 1 \\
0 & \ddots & & \vdots \\
\vdots & \ddots & \ddots & \vdots \\
0 & \cdots & 0 & 1
\end{pmatrix}$$

³⁶The options `renew-dots`, `renew-matrix` can be fixed with the command `\NiceMatrixOptions` like the other options. However, they can also be fixed as options of the command `\usepackage`. There is also a key `transparent` which is an alias for the conjunction of `renew-dots` and `renew-matrix` but it must be considered as obsolete.

10.4 The labels of the dotted lines

The commands `\Ldots`, `\Cdots`, `\Vdots`, `\Ddots`, `\Iddots` and `\Hdotsfor` (and the command `\line` in the `\CodeAfter` which is described p. 28) accept two optional arguments specified by the tokens `_` and `^` for labels positionned below and above the line. The arguments are composed in math mode with `\scriptstyle`.

```


$$\begin{bNiceMatrix} \\ 1 & \hspace*{1cm} & & 0 \\ & \Ddots^{n \text{ times}} & & \\ 0 & & & 1 \\ \end{bNiceMatrix}$$


```

$$\begin{bmatrix} 1 & & & 0 \\ & \ddots^{n \text{ times}} & & \\ 0 & & & 1 \end{bmatrix}$$

10.5 Customisation of the dotted lines

The dotted lines drawn by `\Ldots`, `\Cdots`, `\Vdots`, `\Ddots`, `\Iddots`, `\Hdotsfor` and `\Vdotsfor` (and by the command `\line` in the `\CodeAfter` which is described p. 28) may be customized by three options (specified between square brackets after the command):

- `color`;
- `shorten`;
- `line-style`.

These options may also be fixed with `\NiceMatrixOptions`, as options of `\CodeAfter` or at the level of a given environment but, in those cases, they must be prefixed by `xdots`, and, thus have for names:

- `xdots/color`;
- `xdots/shorten`;
- `xdots/line-style`.

For the clarity of the explanations, we will use those names.

The option `xdots/color`

The option `xdots/color` fixes the color of the dotted line. However, one should remark that the dotted lines drawn in the exterior rows and columns have a special treatment: cf. p. 21.

The option `xdots/shorten`

The option `xdots/shorten` fixes the margin of both extremities of the line. The name is derived from the options “`shorten >`” and “`shorten <`” of Tikz but one should notice that `nicematrix` only provides `xdots/shorten`. The initial value of this parameter is 0.3 em (it is recommended to use a unit of length dependent of the current font).

The option `xdots/line-style`

It should be pointed that, by default, the lines drawn by Tikz with the parameter `dotted` are composed of square dots (and not rounded ones).³⁷

```
\tikz \draw [dotted] (0,0) -- (5,0) ;
```

In order to provide lines with rounded dots in the style of those provided by `\ldots` (at least with the *Computer Modern* fonts), the package `nicematrix` embeds its own system to draw a dotted line (and this system uses PGF and not Tikz). This style is called `standard` and that’s the initial value of the parameter `xdots/line-style`.

However (when Tikz is loaded) it’s possible to use for `xdots/line-style` any style provided by Tikz, that is to say any sequence of options provided by Tikz for the Tikz pathes (with the exception of “`color`”, “`shorten >`” and “`shorten <`”).

³⁷The first reason of this behaviour is that the PDF format includes a description for dashed lines. The lines specified with this descriptor are displayed very efficiently by the PDF readers. It’s easy, starting from these dashed lines, to create a line composed by square dots whereas a line of rounded dots needs a specification of each dot in the PDF file.

Here is for example a tridiagonal matrix with the style `loosely dotted`:

```

 $\begin{pNiceMatrix}[nullify-dots,xdots/line-style=loosely dotted]$ 
a      & b      & 0      & & & \Cdots & 0      & \\
b      & a      & b      & & \Ddots & & & \Vdots \\
0      & b      & a      & & \Ddots & & & \\
      & & \Ddots & & \Ddots & & \Ddots & & 0      & \\
\Vdots & & & & & & & & b      & \\
0      & & \Cdots & & 0      & & b      & & a
\end{pNiceMatrix}
```

$$\begin{pmatrix} a & b & 0 & \cdots & 0 \\ b & a & b & \ddots & \\ 0 & b & a & \ddots & \\ \vdots & \ddots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & b & a \end{pmatrix}$$

10.6 The dotted lines and the rules

The dotted lines determine virtual blocks which have the same behaviour regarding the rules (the rules specified by the specifier `|` in the preamble, by the command `\Hline` and by the keys `hlines`, `vlines`, `hvlines` and `hvlines-except-borders` are not drawn within the blocks).³⁸

```

 $\begin{bNiceMatrix}[margin,hvlines]$ 
\Block{3-3}<\LARGE>\{A\} & & 0 \\
& \hspace*{1cm} & & \Vdots \\
& & 0 \\
0 & \Cdots & 0 & 0
\end{bNiceMatrix}
```

$$\left[\begin{array}{c|c} A & \begin{smallmatrix} 0 \\ \vdots \\ 0 \end{smallmatrix} \\ \hline 0 \cdots 0 & 0 \end{array} \right]$$

11 The `\CodeAfter`

The option `code-after` may be used to give some code that will be executed *after* the construction of the matrix.³⁹

For the legibility of the code, an alternative syntax is provided: it's possible to give the instructions of the `code-after` at the end of the environment, after the keyword `\CodeAfter`. Although `\CodeAfter` is a keyword, it takes in an optional argument (between square brackets). The keys accepted form a subset of the keys of the command `\WithArrowsOptions`.

The experienced users may, for instance, use the PGF/Tikz nodes created by `nicematrix` in the `\CodeAfter`. These nodes are described further beginning on p. 39.

Moreover, several special commands are available in the `\CodeAfter`: `line`, `\SubMatrix`, `\OverBrace` and `\UnderBrace`. We will now present these commands.

11.1 The command `\line` in the `\CodeAfter`

The command `\line` draws directly dotted lines between nodes. It takes in two arguments for the two cells to link, both of the form *i-j* where *i* is the number of the row and *j* is the number of the column. The options available for the customisation of the dotted lines created by `\Cdots`, `\Vdots`, etc. are also available for this command (cf. p. 27).

This command may be used, for example, to draw a dotted line between two adjacent cells.

³⁸On the other side, the command `\line` in the `\CodeAfter` (cf. p. 28) does *not* create block.

³⁹There is also a key `code-before` described p. 14.

```

\NiceMatrixOptions{xdots/shorten = 0.6 em}
\begin{pNiceMatrix}
I      & 0      & & \Cdots & 0      & \\
0      & I      & & \Ddots & \Vdots & \\
\Vdots & \Ddots & & I      & 0      & \\
0      & \Cdots & & 0      & I      & \\
\CodeAfter \line{2-2}{3-3}
\end{pNiceMatrix}

```

$$\begin{pmatrix} I & 0 & \cdots & 0 \\ 0 & I & & \\ \vdots & & \ddots & I \\ 0 & \cdots & 0 & I \end{pmatrix}$$

It can also be used to draw a diagonal line not parallel to the other diagonal lines (by default, the dotted lines drawn by `\Ddots` are “parallelized”: cf. p. 45).

```

\begin{bNiceMatrix}
1      & \Cdots & & 1      & 2      & \Cdots & & 2      & \\
0      & \Ddots & & \Vdots & \Vdots & & \hspace*{2.5cm} & \Vdots & \\
\Vdots & \Ddots & & & & & & & \\
0      & \Cdots & & 0 & 1      & 2      & \Cdots & & 2
\CodeAfter \line[shorten=6pt]{1-5}{4-7}
\end{bNiceMatrix}

```

$$\left[\begin{array}{cc|cc} 1 & \cdots & 1 & 2 \\ 0 & \ddots & \vdots & \vdots \\ \vdots & & \ddots & \vdots \\ 0 & \cdots & 0 & 1 \end{array} \right] \begin{array}{cc|cc} 2 & \cdots & 2 & 2 \\ \vdots & & \ddots & \vdots \\ \vdots & & \ddots & \vdots \\ 2 & \cdots & 2 & 2 \end{array}$$

11.2 The command `\SubMatrix` in the `\CodeAfter`

The command `\SubMatrix` provides a way to put delimiters on a portion of the array considered as a submatrix. The command `\SubMatrix` takes in five arguments:

- the first argument is the left delimiter, which may be any extensible delimiter provided by LaTeX : `(`, `[`, `\{`, `\langle`, `\lgroup`, `\lfloor`, etc. but also the null delimiter `.`;
- the second argument is the upper-left corner of the submatrix with the syntax i - j where i the number of row and j the number of column;
- the third argument is the lower-right corner with the same syntax;
- the fourth argument is the right delimiter;
- the last argument, which is optional, is a list of *key=value* pairs.⁴⁰

One should remark that the command `\SubMatrix` draws the delimiters after the construction of the array: no space is inserted by the command `\SubMatrix` itself. That’s why, in the following example, we have used the key `margin` and you have added by hand some space between the third and fourth column with `@{\hspace{1.5em}}` in the preamble of the array.

```

\[\begin{NiceArray}{ccc@{\hspace{1.5em}}c}[cell-space-limits=2pt,margin]
1      & & 1      & & 1      & & x      & \\
\dfrac{1}{4} & & \dfrac{1}{2} & & \dfrac{1}{4} & & y      & \\
1      & & 2      & & 3      & & z      & \\
\CodeAfter
\SubMatrix({1-1}{3-3})
\SubMatrix({1-4}{3-4})
\end{NiceArray}\]

```

$$\begin{pmatrix} 1 & 1 & 1 \\ \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \\ 1 & 2 & 3 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

⁴⁰There is no optional argument between square brackets in first position because a square bracket just after `\SubMatrix` must be interpreted as the first (mandatory) argument of the command `\SubMatrix`: that bracket is the left delimiter of the sub-matrix to construct (eg.: `\SubMatrix{2-2}{4-7}`).

In fact, the command `\SubMatrix` also takes in two optional arguments specified by the traditional symbols `^` and `_` for material in superscript and subscript.

```

 $\begin{bNiceMatrix}[right-margin=1em]$ 
1 & 1 & 1 \\
1 & a & b \\
1 & c & d \\
\CodeAfter
\SubMatrix[2-2]{3-3}^T
\end{bNiceMatrix}
```

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & a & b \\ 1 & c & d \end{bmatrix}^T$$

The options of the command `\SubMatrix` are as follows:

- `left-xshift` and `right-xshift` shift horizontally the delimiters (there exists also the key `xshift` which fixes both parameters);
- `extra-height` adds a quantity to the total height of the delimiters (height `\ht` + depth `\dp`);
- `delimiters/color` fixes the color of the delimiters (also available in `\NiceMatrixOptions`, in the environments with delimiters and as option of the keyword `\CodeAfter`);
- `slim` is a boolean key: when that key is in force, the horizontal position of the delimiters is computed by using only the contents of the cells of the submatrix whereas, in the general case, the position is computed by taking into account the cells of the whole columns implied in the submatrix (see example below). ;
- `vlines` contents a list of numbers of vertical rules that will be drawn in the sub-matrix (if this key is used without value, all the vertical rules of the sub-matrix are drawn);
- `hlines` is similar to `vlines` but for the horizontal rules;
- `hvlines`, which must be used without value, draws all the vertical and horizontal rules.

One should remark that these keys add their rules after the construction of the main matrix: no space is added between the rows and the columns of the array for theses rules.

All these keys are also available in `\NiceMatrixOptions`, at the level of the environments of `nicematrix` or as option of the command `\CodeAfter` with the prefix `sub-matrix` which means that their names are therefore `sub-matrix/left-xshift`, `sub-matrix/right-xshift`, `sub-matrix/xshift`, etc.

```

 $\begin{NiceArray}{cc@{\hspace{5mm}}l}[cell-space-limits=2pt]$ 
& & \frac{1}{2} \\
& & \frac{1}{4} \\
a & b & \frac{1}{2}a + \frac{1}{4}b \\
c & d & \frac{1}{2}c + \frac{1}{4}d \\
\CodeAfter
\SubMatrix({1-3}{2-3})
\SubMatrix({3-1}{4-2})
\SubMatrix({3-3}{4-3})
\end{NiceArray}
```

$$\begin{pmatrix} \frac{1}{2} \\ \frac{1}{4} \end{pmatrix} \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} \frac{1}{2}a + \frac{1}{4}b \\ \frac{1}{2}c + \frac{1}{4}d \end{pmatrix}$$

Here is the same example with the key `slim` used for one of the submatrices.

```

 $\begin{NiceArray}{cc@{\hspace{5mm}}l}[cell-space-limits=2pt]$ 
& & \frac{1}{2} \\
& & \frac{1}{4} \\
a & b & \frac{1}{2}a + \frac{1}{4}b \\
c & d & \frac{1}{2}c + \frac{1}{4}d \\
\CodeAfter
\SubMatrix({1-3}{2-3})[slim]
\SubMatrix({3-1}{4-2})
\SubMatrix({3-3}{4-3})
\end{NiceArray}
```

$$\begin{pmatrix} \frac{1}{2} \\ \frac{1}{4} \end{pmatrix} \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} \frac{1}{2}a + \frac{1}{4}b \\ \frac{1}{2}c + \frac{1}{4}d \end{pmatrix}$$

There is also a key `name` which gives a name to the submatrix created by `\SubMatrix`. That name is used to create PGF/Tikz nodes: cf p. 43.

It's also possible to specify some delimiters⁴¹ by placing them in the preamble of the environment (for the environments with a preamble: `{NiceArray}`, `{pNiceArray}`, etc.). This syntax is inspired by the extension `blkarray`.

When there are two successive delimiters (necessarily a closing one following by an opening one for another submatrix), a space equal to `\enskip` is automatically inserted.

```

 $\begin{pNiceArray}{(c)(c)(c)}
a_{11} & a_{12} & & a_{13} \\
a_{21} & \displaystyle \int_0^1 \frac{1}{x^2+1} dx & & a_{23} \\
a_{31} & a_{32} & & a_{33}
\end{pNiceArray}$ 

```

$$\begin{pmatrix} a_{11} \\ a_{21} \\ a_{31} \end{pmatrix} \begin{pmatrix} a_{12} \\ \int_0^1 \frac{1}{x^2+1} dx \\ a_{32} \end{pmatrix} \begin{pmatrix} a_{13} \\ a_{23} \\ a_{33} \end{pmatrix}$$

11.3 The commands `\OverBrace` and `\UnderBrace` in the `\CodeAfter`

New 6.4

The commands `\OverBrace` and `\UnderBrace` provide a way to put horizontal braces on a part of the array. These commands take in three arguments:

- the first argument is the upper-left corner of the submatrix with the syntax i - j where i the number of row and j the number of column;
- the second argument is the lower-right corner with the same syntax;
- the third argument is the label of the brace that will be put by `nicematrix` (with PGF) above the brace (for the command `\OverBrace`) or under the brace (for `\UnderBrace`).

```

\begin{pNiceMatrix}
1 & 2 & 3 & 4 & 5 & 6 \\
11 & 12 & 13 & 14 & 15 & 16 \\
\CodeAfter
\OverBrace{1-1}{2-3}{A}
\OverBrace{1-4}{2-6}{B}
\end{pNiceMatrix}

```

$$\begin{pmatrix} \overbrace{1 \ 2 \ 3}^A & \overbrace{4 \ 5 \ 6}^B \\ 11 & 12 & 13 & 14 & 15 & 16 \end{pmatrix}$$

In fact, the commands `\OverBrace` and `\UnderBrace` take in an optional argument (in first position and between square brackets) for a list of `key=value` pairs. The available keys are:

- `left-shorten` and `right-shorten` which do not take in value; when the key `left-shorten` is used, the abscissa of the left extremity of the brace is computed with the contents of the cells of the involved sub-array, otherwise, the position of the potential vertical rule is used (idem for `right-shorten`).
- `shorten`, which is the conjunction of the keys `left-shorten` and `right-shorten`;
- `yshift`, which shifts vertically the brace (and its label).

⁴¹Those delimiters are `(`, `[`, `\{` and the closing ones. Of course, it's also possible to put `|` and `||` in the preamble of the environment.

```

\begin{pNiceMatrix}
1 & 2 & 3 & 4 & 5 & 6 & \\
11 & 12 & 13 & 14 & 15 & 16 & \\
\CodeAfter
  \OverBrace[shorten,yshift=3pt]{1-1}{2-3}{A}
  \OverBrace[shorten,yshift=3pt]{1-4}{2-6}{B}
\end{pNiceMatrix}

```

$$\begin{pmatrix}
 \overbrace{1 \quad 2 \quad 3}^A & \overbrace{4 \quad 5 \quad 6}^B \\
 11 \quad 12 \quad 13 & 14 \quad 15 \quad 16
 \end{pmatrix}$$

12 The notes in the tabulars

12.1 The footnotes

The package `nicematrix` allows, by using `footnote` or `footnotehyper`, the extraction of the notes inserted by `\footnote` in the environments of `nicematrix` and their composition in the footpage with the other notes of the document.

If `nicematrix` is loaded with the option `footnote` (with `\usepackage[footnote]{nicematrix}` or with `\PassOptionsToPackage`), the package `footnote` is loaded (if it is not yet loaded) and it is used to extract the footnotes.

If `nicematrix` is loaded with the option `footnotehyper`, the package `footnotehyper` is loaded (if it is not yet loaded) and it is used to extract footnotes.

Caution: The packages `footnote` and `footnotehyper` are incompatible. The package `footnotehyper` is the successor of the package `footnote` and should be used preferently. The package `footnote` has some drawbacks, in particular: it must be loaded after the package `xcolor` and it is not perfectly compatible with `hyperref`.

12.2 The notes of tabular

The package `nicematrix` also provides a command `\tabularnote` which gives the ability to specify notes that will be composed at the end of the array with a width of line equal to the width of the array (excepted the potential exterior columns). With no surprise, that command is available only in the environments without delimiters, that is to say `{NiceTabular}`, `{NiceArray}` and `{NiceMatrix}`. In fact, this command is available only if the extension `enumitem` has been loaded (before or after `nicematrix`). Indeed, the notes are composed at the end of the array with a type of list provided by the package `enumitem`.

```

\begin{NiceTabular}{@{}llr@{}}
\toprule \RowStyle{\bfseries}
Last name & First name & Birth day \\
\midrule
Achard\tabularnote{Achard is an old family of the Poitou.}
& Jacques & 5 juin 1962 \\
Lefebvre\tabularnote{The name Lefebvre is an alteration of the name Lefebure.}
& Mathilde & 23 mai 1988 \\
Vanesse & Stephany & 30 octobre 1994 \\
Dupont & Chantal & 15 janvier 1998 \\
\bottomrule
\end{NiceTabular}

```


Last name	First name	Birth day
Achard ^a	Jacques	June 5, 2005
Lefebvre ^b	Mathilde	January 23, 1975
Vanesse	Stephany	October 30, 1994
Dupont	Chantal	January 15, 1998

^a Achard is an old family of the Poitou.

^b The name Lefebvre is an alteration of the name Lefebure.

- If you have several successive commands `\tabularnote{...}` with no space at all between them, the labels of the corresponding notes are composed together, separated by commas (this is similar to the option `multiple` of `footmisc` for the footnotes).
- If a command `\tabularnote{...}` is exactly at the end of a cell (with no space at all after), the label of the note is composed in an overlapping position (towards the right). This structure may provide a better alignment of the cells of a given column.
- If the key `notes/para` is used, the notes are composed at the end of the array in a single paragraph (as with the key `para` of `threeparttable`).
- There is a key `tabularnote` which provides a way to insert some text in the zone of the notes before the numbered tabular notes.
- If the package `booktabs` has been loaded (before or after `nicematrix`), the key `notes/bottomrule` draws a `\bottomrule` of `booktabs` after the notes.
- The command `\tabularnote` may be used before the environment of `nicematrix`. Thus, it's possible to use it on the title inserted by `\caption` in an environment `{table}` of LaTeX.
- It's possible to create a reference to a tabular note created by `\tabularnote` (with the usual command `\label` used after the `\tabularnote`).

For an illustration of some of those remarks, see table 1, p. 34. This table has been composed with the following code.

```
\begin{table}
\setlength{\belowcaptionskip}{1ex}
\centering
\caption{Use of \texttt{\textbackslash tabularnote}\tabularnote{It's possible
to put a note in the caption.}}
\label{t:tabularnote}
\begin{NiceTabular}{@{}llc@{}}
[notes/bottomrule, tabularnote = Some text before the notes.]
\toprule
Last name & First name & Length of life \\
\midrule
Churchill & Wiston & 91\\
Nightingale\tabularnote{Considered as the first nurse of
history.}\tabularnote{Nicknamed ``the Lady with the Lamp''.}
& Florence & 90 \\
Schoelcher & Victor & 89\tabularnote{The label of the note is overlapping.}\\
Touchet & Marie & 89 \\
Wallis & John & 87 \\
\bottomrule
\end{NiceTabular}
\end{table}
```

12.3 Customisation of the tabular notes

The tabular notes can be customized with a set of keys available in `\NiceMatrixOptions`. The name of these keys is prefixed by `notes`.

- `notes/para`
- `notes/bottomrule`
- `notes/style`
- `notes/label-in-tabular`
- `notes/label-in-list`
- `notes/enumitem-keys`
- `notes/enumitem-keys-para`
- `notes/code-before`

For sake of commodity, it is also possible to set these keys in `\NiceMatrixOptions` via a key `notes` which takes in as value a list of pairs `key=value` where the name of the keys need no longer be prefixed by `notes`:

```
\NiceMatrixOptions
{
  notes =
  {
    bottomrule ,
    style = ... ,
    label-in-tabular = ... ,
    enumitem-keys =
    {
      labelsep = ... ,
      align = ... ,
      ...
    }
  }
}
```

We detail these keys.

- The key `notes/para` requires the composition of the notes (at the end of the tabular) in a single paragraph.

Initial value: `false`

That key is also available within a given environment.

Table 1: Use of `\tablarnote`^a

Last name	First name	Length of life
Churchill	Wiston	91
Nightingale ^{b,c}	Florence	90
Schoelcher	Victor	89 ^d
Touchet	Marie	89
Wallis	John	87

Some text before the notes.

^a It's possible to put a note in the caption.

^b Considered as the first nurse of history.

^c Nicknamed “the Lady with the Lamp”.

^d The label of the note is overlapping.

- The key `notes/bottomrule` adds a `\bottomrule` of `booktabs` *after* the notes. Of course, that rule is drawn only if there is really notes in the tabular. The package `booktabs` must have been loaded (before or after the package `nicematrix`). If it is not, an error is raised.

Initial value: `false`

That key is also available within a given environment.

- The key `notes/style` is a command whose argument is specified by `#1` and which gives the style of numerotation of the notes. That style will be used by `\ref` when referencing a tabular note marked with a command `\label`. The labels formatted by that style are used, separated by commas, when the user puts several consecutive commands `\tabularnote`. The marker `#1` is meant to be the name of a LaTeX counter.

Initial value: `\textit{\alph{#1}}`

Another possible value should be a mere `\arabic{#1}`

- The key `notes/label-in-tabular` is a command whose argument is specified by `#1` which is used when formatting the label of a note in the tabular. Internally, this number of note has already been formatted by `notes/style` before sent to that command.

Initial value: `#1`

In French, it's a tradition of putting a small space before the label of note. That tuning could be achieved by the following code:

```
\NiceMatrixOptions{notes/label-in-tabular = \,\textsuperscript{#1}}
```

- The key `notes/label-in-list` is a command whose argument is specified by `#1` which is used when formatting the label in the list of notes at the end of the tabular. Internally, this number of note has already been formatted by `notes/style` before sent to that command.

Initial value: `#1`

In French, the labels of notes are not composed in upper position when composing the notes. Such behaviour could be achieved by:

```
\NiceMatrixOptions{notes/label-in-list = #1.\nobreak\hspace{0.25em}}
```

The command `\nobreak` is for the event that the option `para` is used.

- The notes are composed at the end of the tabular by using internally a style of list of `enumitem`. This style of list is defined as follows (with, of course, keys of `enumitem`):

```
noitemsep , leftmargin = * , align = left , labelsep = 0pt
```

The specification `align = left` in that style requires a composition of the label leftwards in the box affected to that label. With that tuning, the notes are composed flush left, which is pleasant when composing tabulars in the spirit of `booktabs` (see for example the table 1, p. 34).

The key `notes/enumitem-keys` specifies a list of pairs `key=value` (following the specifications of `enumitem`) to customize that style of list (it uses internally the command `\setlist*` of `enumitem`).

- The key `notes/enumitem-keys-para` is similar to the previous one but corresponds to the type of list used when the option `para` is in force. Of course, when the option `para` is used, a list of type `inline` (as called by `enumitem`) is used and the pairs `key=value` should correspond to such a list of type `inline`.

Initially, the style of list is defined by: `afterlabel = \nobreak, itemjoin = \quad`

- The key `notes/code-before` is a token list inserted by `nicematrix` just before the composition of the notes at the end of the tabular.

Initial value: `empty`

For example, if one wishes to compose all the notes in gray and `\footnotesize`, he should use that key:

```
\NiceMatrixOptions{notes/code-before = \footnotesize \color{gray}}
```

It's also possible to add `\raggedright` or `\RaggedRight` in that key (`\RaggedRight` is a command of `ragged2e`).

For an example of customisation of the tabular notes, see p. 48.

12.4 Use of `{NiceTabular}` with `threeparttable`

If you wish to use the environment `{NiceTabular}`, `{NiceTabular*}` `{NiceTabularX}` in an environment `{threeparttable}` of the eponymous package, you have to patch the environment `{threeparttable}` with the following code (with a version of LaTeX at least 2020/10/01).

```
\makeatletter
\AddToHook{env/threeparttable/begin}
{
  \TPT@hookin{NiceTabular}\TPT@hookin{NiceTabular*}\TPT@hookin{NiceTabularX}
}
\makeatother
```

13 Other features

13.1 Use of the column type `S` of `siunitx`

If the package `siunitx` is loaded (before or after `nicematrix`), it's possible to use the `S` column type of `siunitx` in the environments of `nicematrix`. The implementation doesn't use explicitly any private macro of `siunitx`.

```
$\begin{pNiceArray}{ScWc{1cm}c}[nullify-dots,first-row]
{C_1} & \Cdots & & C_n \\
2.3 & 0 & \Cdots & 0 \\
12.4 & \Vdots & & \Vdots \\
1.45 & \\
7.2 & 0 & \Cdots & 0 \\
\end{pNiceArray}$
```

$$\begin{pmatrix} C_1 & \dots & C_n \\ 2.3 & 0 & \dots & 0 \\ 12.4 & \vdots & & \vdots \\ 1.45 & \vdots & & \vdots \\ 7.2 & 0 & \dots & 0 \end{pmatrix}$$

On the other hand, the `d` columns of the package `dcolumn` are not supported by `nicematrix`.

13.2 Alignment option in `{NiceMatrix}`

The environments without preamble (`{NiceMatrix}`, `{pNiceMatrix}`, `{bNiceMatrix}`, etc.) provide two options `l` and `r` which generate all the columns aligned leftwards (or rightwards).

```
$\begin{bNiceMatrix}[r]
\cos x & - \sin x \\
\sin x & \cos x
\end{bNiceMatrix}$
```

$$\begin{bmatrix} \cos x & -\sin x \\ \sin x & \cos x \end{bmatrix}$$

13.3 The command `\rotate`

The package `nicematrix` provides a command `\rotate`. When used in the beginning of a cell, this command composes the contents of the cell after a rotation of 90° in the direct sens.

In the following command, we use that command in the `code-for-first-row`.⁴²

⁴²It can also be used in `\RowStyle` (cf. p. 18).

```

\NiceMatrixOptions%
{code-for-first-row = \scriptstyle \rotate \text{image of },
code-for-last-col = \scriptstyle }
$A = \begin{pNiceMatrix}[first-row,last-col=4]
e_1 & e_2 & e_3 & \\
1 & 2 & 3 & e_1 \\
4 & 5 & 6 & e_2 \\
7 & 8 & 9 & e_3
\end{pNiceMatrix}$

```

$$A = \begin{pmatrix} \text{image of } e_1 & \text{image of } e_2 & \text{image of } e_3 \\ 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} \begin{matrix} e_1 \\ e_2 \\ e_3 \end{matrix}$$

If the command `\rotate` is used in the “last row” (exterior to the matrix), the corresponding elements are aligned upwards as shown below.

```

\NiceMatrixOptions%
{code-for-last-row = \scriptstyle \rotate ,
code-for-last-col = \scriptstyle }
$A = \begin{pNiceMatrix}[last-row=4,last-col=4]
1 & 2 & 3 & e_1 \\
4 & 5 & 6 & e_2 \\
7 & 8 & 9 & e_3 \\
\text{image of } e_1 & \text{image of } e_2 & \text{image of } e_3 & 
\end{pNiceMatrix}$

```

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} \begin{matrix} e_1 \\ e_2 \\ e_3 \end{matrix}$$

13.4 The option `small`

With the option `small`, the environments of the package `nicematrix` are composed in a way similar to the environment `{smallmatrix}` of the package `amsmath` (and the environments `{psmallmatrix}`, `{bsmallmatrix}`, etc. of the package `mathtools`).

```

$\begin{bNiceArray}{cccc|c}[small,
last-col,
code-for-last-col = \scriptscriptstyle,
columns-width = 3mm ]
1 & -2 & 3 & 4 & 5 \\
0 & 3 & 2 & 1 & 2 & L_2 \text{ \gets } 2L_1 - L_2 \\
0 & 1 & 1 & 2 & 3 & L_3 \text{ \gets } L_1 + L_3
\end{bNiceArray}$

```

$$\left[\begin{array}{cccc|c} 1 & -2 & 3 & 4 & 5 \\ 0 & 3 & 2 & 1 & 2 \\ 0 & 1 & 1 & 2 & 3 \end{array} \right] \begin{matrix} \\ L_2 \leftarrow 2L_1 - L_2 \\ L_3 \leftarrow L_1 + L_3 \end{matrix}$$

One should note that the environment `{NiceMatrix}` with the option `small` is not composed *exactly* as the environment `{smallmatrix}`. Indeed, all the environments of `nicematrix` are constructed upon `{array}` (of the package `array`) whereas the environment `{smallmatrix}` is constructed directly with an `\halign` of TeX.

In fact, the option `small` corresponds to the following tuning:

- the cells of the array are composed with `\scriptstyle`;
- `\arraystretch` is set to 0.47;
- `\arraycolsep` is set to 1.45 pt;
- the characteristics of the dotted lines are also modified.

13.5 The counters iRow and jCol

In the cells of the array, it's possible to use the LaTeX counters `iRow` and `jCol` which represent the number of the current row and the number of the current column⁴³. Of course, the user must not change the value of these counters which are used internally by `nicematrix`.

In the `\CodeBefore` (cf. p. 14) and in the `\CodeAfter` (cf. p. 28), `iRow` represents the total number of rows (excepted the potential exterior rows) and `jCol` represents the total number of columns (excepted the potential exterior columns).

```

 $\begin{pNiceMatrix}$ % don't forget the %
    [first-row,
     first-col,
     code-for-first-row = \mathbf{\alpha{jCol}} ,
     code-for-first-col = \mathbf{\arabic{iRow}} ]
& & & & \\
& 1 & 2 & 3 & 4 \\
& 5 & 6 & 7 & 8 \\
& 9 & 10 & 11 & 12
\end{pNiceMatrix}

```

$$\begin{matrix} & \mathbf{a} & \mathbf{b} & \mathbf{c} & \mathbf{d} \\ \mathbf{1} & 1 & 2 & 3 & 4 \\ \mathbf{2} & 5 & 6 & 7 & 8 \\ \mathbf{3} & 9 & 10 & 11 & 12 \end{matrix}$$

If LaTeX counters called `iRow` and `jCol` are defined in the document by packages other than `nicematrix` (or by the final user), they are shadowed in the environments of `nicematrix`.

The package `nicematrix` also provides commands in order to compose automatically matrices from a general pattern. These commands are `\AutoNiceMatrix`, `\pAutoNiceMatrix`, `\bAutoNiceMatrix`, `\vAutoNiceMatrix`, `\VAutoNiceMatrix` and `\BAutoNiceMatrix`.

These commands take in two mandatory arguments. The first is the format of the matrix, with the syntax `n-p` where `n` is the number of rows and `p` the number of columns. The second argument is the pattern (it's a list of tokens which are inserted in each cell of the constructed matrix).

```

 $C = \pAutoNiceMatrix{3-3}{C_{\arabic{iRow},\arabic{jCol}}}$ 

```

$$C = \begin{pmatrix} C_{1,1} & C_{1,2} & C_{1,3} \\ C_{2,1} & C_{2,2} & C_{2,3} \\ C_{3,1} & C_{3,2} & C_{3,3} \end{pmatrix}$$

13.6 The option light-syntax

The option `light-syntax` (inpired by the package `spalign`) allows the user to compose the arrays with a lighter syntax, which gives a better legibility of the TeX source.

When this option is used, one should use the semicolon for the end of a row and spaces or tabulations to separate the columns. However, as usual in the TeX world, the spaces after a control sequence are discarded and the elements between curly braces are considered as a whole.

```

 $\begin{bNiceMatrix}[light-syntax,first-row,first-col]$ 
{} a          b          ;
a 2\cos a     {\cos a + \cos b} ;
b \cos a+\cos b { 2 \cos b }
\end{bNiceMatrix}

```

$$\begin{matrix} & a & b \\ a & 2 \cos a & \cos a + \cos b \\ b & \cos a + \cos b & 2 \cos b \end{matrix}$$

It's possible to change the character used to mark the end of rows with the option `end-of-row`. As said before, the initial value is a semicolon.

When the option `light-syntax` is used, it is not possible to put verbatim material (for example with the command `\verb`) in the cells of the array.⁴⁴

⁴³We recall that the exterior "first row" (if it exists) has the number 0 and that the exterior "first column" (if it exists) has also the number 0.

⁴⁴The reason is that, when the option `light-syntax` is used, the whole content of the environment is loaded as a TeX argument to be analyzed. The environment doesn't behave in that case as a standard environment of LaTeX which only put TeX commands before and after the content.

13.7 Color of the delimiters

For the environments with delimiters (`{pNiceArray}`, `{pNiceMatrix}`, etc.), it's possible to change the color of the delimiters with the key `delimiters/color`.

```

\begin{bNiceMatrix}[delimiters/color=red]
1 & 2 & \\
3 & 4 & 
\end{bNiceMatrix}
```

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

This colour also applies to the delimiters drawn by the command `\SubMatrix` (cf. p. 29).

13.8 The environment `{NiceArrayWithDelims}`

In fact, the environment `{pNiceArray}` and its variants are based upon a more general environment, called `{NiceArrayWithDelims}`. The first two mandatory arguments of this environment are the left and right delimiters used in the construction of the matrix. It's possible to use `{NiceArrayWithDelims}` if we want to use atypical or asymmetrical delimiters.

```

\begin{NiceArrayWithDelims}
  {\downarrow}{\uparrow}{ccc}[margin]
1 & 2 & 3 \\
4 & 5 & 6 \\
7 & 8 & 9 
\end{NiceArrayWithDelims}
```

$$\begin{array}{ccc} \downarrow & 1 & 2 & 3 & \uparrow \\ & 4 & 5 & 6 & \\ & 7 & 8 & 9 & \end{array}$$

14 Use of Tikz with nicematrix

14.1 The nodes corresponding to the contents of the cells

The package `nicematrix` creates a PGF/Tikz node for each (non-empty) cell of the considered array. These nodes are used to draw the dotted lines between the cells of the matrix (inter alia).

Caution : By default, no node is created in a empty cell.

However, it's possible to impose the creation of a node with the command `\NotEmpty`. ⁴⁵

The nodes of a document must have distinct names. That's why the names of the nodes created by `nicematrix` contains the number of the current environment. Indeed, the environments of `nicematrix` are numbered by a internal global counter.

In the environment with the number n , the node of the row i and column j has for name `nm-n-i-j`. The command `\NiceMatrixLastEnv` provides the number of the last environment of `nicematrix` (for LaTeX, it's a “fully expandable” command and not a counter).

However, it's advisable to use instead the key `name`. This key gives a name to the current environment. When the environment has a name, the nodes are accessible with the name “*name-i-j*” where *name* is the name given to the array and i and j the numbers of row and column. It's possible to use these nodes with PGF but the final user will probably prefer to use Tikz (which is a convenient layer upon PGF). However, one should remind that `nicematrix` doesn't load Tikz by default. In the following examples, we assume that Tikz has been loaded.

```

\begin{pNiceMatrix}[name=mymatrix]
1 & 2 & 3 \\
4 & 5 & 6 \\
7 & 8 & 9 
\end{pNiceMatrix}
\tikz[remember picture,overlay]
  \draw (mymatrix-2-2) circle (2mm) ;
```

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & \textcircled{5} & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

⁴⁵One should note that, with that command, the cell is considered as non-empty, which has consequences for the continuous dotted lines (cf. p. 23) and the computation of the “corners” (cf. p. 10).

Don't forget the options `remember picture` and `overlay`.

In the `\CodeAfter`, the things are easier : one must refer to the nodes with the form $i-j$ (we don't have to indicate the environment which is of course the current environment).

```

 $\begin{pNiceMatrix}$ 
1 & 2 & 3 \\
4 & 5 & 6 \\
7 & 8 & 9
 $\CodeAfter$ 
\tikz \draw (2-2) circle (2mm) ;
 $\end{pNiceMatrix}$ 

```

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & \textcircled{5} & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

In the following example, we have underlined all the nodes of the matrix (we explain below the technic used : cf. p. 54).

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

New 6.3 The nodes of the last column (excepted the potential «last column» specified by `last-col`) may also be indicated by i -`last`. Similarly, the nodes of the last row may be indicated by `last`- j .

14.1.1 The columns V of varwidth

When the extension `varwidth` is loaded, the columns of the type `V` defined by `varwidth` are supported by `nicematrix`. It may be interessant to notice that, for a cell of a column of type `V`, the PGF/Tikz node created by `nicematrix` for the content of that cell has a width adjusted to the content of the cell. This is in contrast to the case of the columns of type `p`, `m` or `b` for which the nodes have always a width equal to the width of the column. In the following example, the command `\lipsum` is provided by the eponymous package.

```

 $\begin{NiceTabular}\{V\{10cm\}\}$ 
\bfseries \large
Titre \\
\lipsum[1][1-4]
 $\CodeAfter$ 
\tikz \draw [rounded corners] (1-1) -| (last-|2) -- (last-|1) |- (1-1) ;
 $\end{NiceTabular}$ 

```

Titre

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur id, vulputate a, magna.

We have used the nodes corresponding to the position of the potential rules, which are described below (cf. p. 42).

14.2 The “medium nodes” and the “large nodes”

In fact, the package `nicematrix` can create “extra nodes”: the “medium nodes” and the “large nodes”. The first ones are created with the option `create-medium-nodes` and the second ones with the option `create-large-nodes`.⁴⁶

These nodes are not used by `nicematrix` by default, and that's why they are not created by default.

⁴⁶There is also an option `create-extra-nodes` which is an alias for the conjunction of `create-medium-nodes` and `create-large-nodes`.

The names of the “medium nodes” are constructed by adding the suffix “-medium” to the names of the “normal nodes”. In the following example, we have underlined the “medium nodes”. We consider that this example is self-explanatory.

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

The names of the “large nodes” are constructed by adding the suffix “-large” to the names of the “normal nodes”. In the following example, we have underlined the “large nodes”. We consider that this example is self-explanatory.⁴⁷

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

The “large nodes” of the first column and last column may appear too small for some usage. That’s why it’s possible to use the options `left-margin` and `right-margin` to add space on both sides of the array and also space in the “large nodes” of the first column and last column. In the following example, we have used the options `left-margin` and `right-margin`.⁴⁸

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

It’s also possible to add more space on both side of the array with the options `extra-left-margin` and `extra-right-margin`. These margins are not incorporated in the “large nodes”. It’s possible to fix both values with the option `extra-margin` and, in the following example, we use `extra-margin` with the value 3 pt.

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

Be careful : These nodes are reconstructed from the contents of the contents cells of the array. Usually, they do not correspond to the cells delimited by the rules (if we consider that these rules are drawn).

Here is an array composed with the following code:

```
\large
\begin{NiceTabular}{wl{2cm}ll}[hvlines]
fraise & amande & abricot \\
prune & pêche & poire \\
noix & noisette & brugnol
\end{NiceTabular}
```

fraise	amande	abricot
prune	pêche	poire
noix	noisette	brugnol

Here, we have colored all the cells of the array with `\chessboardcolors`.

fraise	amande	abricot
prune	pêche	poire
noix	noisette	brugnol

⁴⁷There is no “large nodes” created in the exterior rows and columns (for these rows and columns, cf. p. 21).

⁴⁸The options `left-margin` and `right-margin` take dimensions as values but, if no value is given, the default value is used, which is `\arraycolsep` (by default: 5 pt). There is also an option `margin` to fix both `left-margin` and `right-margin` to the same value.

Here are the “large nodes” of this array (without use of `margin` nor `extra-margin`).

fraise	amande	abricot
prune	pêche	poire
noix	noisette	brugnon

The nodes we have described are not available by default in the `\CodeBefore` (described p. 14). It’s possible to have these nodes available in the `\CodeBefore` by using the key `create-cell-nodes` of the keyword `\CodeBefore` (in that case, the nodes are created first before the construction of the array by using informations written on the `aux` file and created a second time during the construction of the array itself).

Here is an example which uses these nodes in the `\CodeAfter`.

```
\begin{NiceArray}{c@{\;}c@{\;}c@{\;}c}[create-medium-nodes]
  u_1 & - & u_0 & = & r & \\
  u_2 & - & u_1 & = & r & \\
  u_3 & - & u_2 & = & r & \\
  u_4 & - & u_3 & = & r & \\
  \phantom{u_5} & & \phantom{u_4} & & \smash{\vdots} & \\
  u_n & - & u_{n-1} & = & r & \\
\hline
  u_n & - & u_0 & = & nr & \\
\CodeAfter
  \tikz[very thick, red, opacity=0.4,name suffix = -medium]
  \draw (1-1.north west) -- (2-3.south east)
  (2-1.north west) -- (3-3.south east)
  (3-1.north west) -- (4-3.south east)
  (4-1.north west) -- (5-3.south east)
  (5-1.north west) -- (6-3.south east) ;
\end{NiceArray}
```

$$\begin{array}{rcl}
 u_1 - u_0 & = & r \\
 u_2 - u_1 & = & r \\
 u_3 - u_2 & = & r \\
 u_4 - u_3 & = & r \\
 & \vdots & \\
 u_n - u_{n-1} & = & r \\
 \hline
 u_n - u_0 & = & nr
 \end{array}$$

14.3 The nodes which indicate the position of the rules

The package `nicematrix` creates a PGF/Tikz node merely called i (with the classical prefix) at the intersection of the horizontal rule of number i and the vertical rule of number i (more specifically the potential position of those rules because maybe there are not actually drawn). The last node has also an alias called `last`. There is also a node called $i.5$ midway between the node i and the node $i + 1$. These nodes are available in the `\CodeBefore` and the `\CodeAfter`.

	$\bullet^{1.5}$	tulipe	lys
arum		$\bullet^{2.5}$	violette mauve
muguet	dahlia		$\bullet^{3.5}$

If we use Tikz (we remind that `nicematrix` does not load Tikz by default, by only PGF, which is a sub-layer of Tikz), we can access, in the `\CodeAfter` but also in the `\CodeBefore`, to the intersection of the (potential) horizontal rule i and the (potential) vertical rule j with the syntax $(i-j)$.

```

\begin{NiceMatrix}
\CodeBefore
\tikz \draw [fill=red!15] (7-|4) |- (8-|5) |- (9-|6) |- cycle ;
\Body
1 \\\
1 & 1 \\\
1 & 2 & 1 \\\
1 & 3 & 3 & 1 \\\
1 & 4 & 6 & 4 & 1 \\\
1 & 5 & 10 & 10 & 5 & 1 \\\
1 & 6 & 15 & 20 & 15 & 6 & 1 \\\
1 & 7 & 21 & 35 & 35 & 21 & 7 & 1 \\\
1 & 8 & 28 & 56 & 70 & 56 & 28 & 8 & 1
\end{NiceMatrix}

```

```

1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
1 6 15 20 15 6 1
1 7 21 35 35 21 7 1
1 8 28 56 70 56 28 8 1

```

The nodes of the form $i.5$ may be used, for example to cross a row of a matrix (if Tikz is loaded).

```

$\begin{pNiceArray}{ccc|c}
2 & 1 & 3 & 0 \\\
3 & 3 & 1 & 0 \\\
3 & 3 & 1 & 0
\CodeAfter
\tikz \draw [red] (3.5-|1) -- (3.5-|last) ;
\end{pNiceArray}$

```

$$\left(\begin{array}{ccc|c} 2 & 1 & 3 & 0 \\ 3 & 3 & 1 & 0 \\ \hline 3 & 3 & 1 & 0 \end{array}\right)$$

14.4 The nodes corresponding to the command `\SubMatrix`

The command `\SubMatrix` available in the `\CodeAfter` has been described p. 29.

If a command `\SubMatrix` has been used with the key `name` with an expression such as `name=MyName` three PGF/Tikz nodes are created with the names `MyName-left`, `MyName` and `MyName-right`.

The nodes `MyName-left` and `MyName-right` correspond to the delimiters left and right and the node `MyName` correspond to the submatrix itself.

In the following example, we have highlighted these nodes (the submatrix itself has been created with `\SubMatrix\{{2-2}\{3-3}\}`).

$$\left(\begin{array}{cccc} 121 & 23 & 345 & 345 \\ 45 & \left\{ \begin{array}{cc} 346 & 863 \end{array} \right\} & 444 & \\ 3462 & \left\{ \begin{array}{cc} 38458 & 34 \end{array} \right\} & 294 & \\ 34 & 7 & 78 & 309 \end{array}\right)$$

15 API for the developpers

The package `nicematrix` provides two variables which are internal but public⁴⁹:

- `\g_nicematrix_code_before_tl` ;
- `\g_nicematrix_code_after_tl`.

These variables contain the code of what we have called the “code-before” (usually specified at the beginning of the environment with the syntax using the keywords `\CodeBefore` and `\Body`) and the “code-after” (usually specified at the end of the environment after the keyword `\CodeAfter`). The developer can use them to add code from a cell of the array (the affectation must be global, allowing to exit the cell, which is a TeX group).

One should remark that the use of `\g_nicematrix_code_before_tl` needs one compilation more (because the instructions are written on the `aux` file to be used during the next run).

Example : We want to write a command `\crossbox` to draw a cross in the current cell. This command will take in an optional argument between square brackets for a list of pairs *key-value* which will be given to Tikz before the drawing.

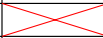
It’s possible to program such command `\crossbox` as follows, explicitly using the public variable `\g_nicematrix_code_after_tl`.

```
\ExplSyntaxOn
\cs_new_protected:Nn \__pantigny_crossbox:nnn
{
  \tikz \draw [ #3 ]
    ( #1 -| \int_eval:n { #2 + 1 } ) -- ( \int_eval:n { #1 + 1 } -| #2 )
    ( #1 -| #2 ) -- ( \int_eval:n { #1 + 1 } -| \int_eval:n { #2 + 1 } ) ;
}

\NewDocumentCommand \crossbox { ! 0 { } }
{
  \tl_gput_right:Nx \g_nicematrix_code_after_tl
  {
    \__pantigny_crossbox:nnn
    { \int_use:c { c@iRow } }
    { \int_use:c { c@jCol } }
    { \exp_not:n { #1 } }
  }
}
\ExplSyntaxOff
```

Here is an example of utilisation:

```
\begin{NiceTabular}{ccc}[hvlines]
merlan & requin & cabillaud \\
baleine & \crossbox[red] & morue \\
mante & raie & poule
\end{NiceTabular}
```

merlan	requin	cabillaud
baleine		morue
mante	raie	poule

⁴⁹According to the LaTeX3 conventions, each variable with name beginning with `\g_nicematrix` ou `\l_nicematrix` is public and each variable with name beginning with `\g__nicematrix` or `\l__nicematrix` is private.

16 Technical remarks

16.1 Definition of new column types

The package `nicematrix` provides the command `\OnlyMainNiceMatrix` which is meant to be used in definitions of new column types. Its argument is evaluated if and only if we are in the main part of the array, that is to say not in a potential exterior row.

For example, one may wish to define a new column type `?` in order to draw a (black) heavy rule of width 1 pt. The following definition will do the job⁵⁰:

```
\newcolumnntype{?}{\OnlyMainNiceMatrix{\vrule width 1 pt}}
```

The heavy vertical rule won't extend in the exterior rows.⁵¹

```
\begin{pNiceArray}{cc?cc}[first-row,last-row=3]
```

```
C_1 & C_2 & C_3 & C_4 \\\
```

```
a & b & c & d \\\
```

```
e & f & g & h \\\
```

```
C_1 & C_2 & C_3 & C_4
```

```
\end{pNiceArray}$
```

$$\begin{array}{cc|cc} C_1 & C_2 & C_3 & C_4 \\ \hline a & b & c & d \\ e & f & g & h \\ C_1 & C_2 & C_3 & C_4 \end{array}$$

This specifier `?` may be used in the standard environments `{tabular}` and `{array}` (of the package `array`) and, in this case, the command `\OnlyMainNiceMatrix` is no-op.

16.2 Diagonal lines

By default, all the diagonal lines⁵² of a same array are “parallelized”. That means that the first diagonal line is drawn and, then, the other lines are drawn parallel to the first one (by rotation around the left-most extremity of the line). That's why the position of the instructions `\Ddots` in the array can have a marked effect on the final result.

In the following examples, the first `\Ddots` instruction is written in color:

Example with parallelization (default):

```
$A = \begin{pNiceMatrix}
1      & \Cdots &      & 1      \\\
a+b    & \Ddots &      & \Vdots \\\
\Vdots & \Ddots &      & \\\
a+b    & \Cdots & a+b  & 1
\end{pNiceMatrix}$
```

$$A = \begin{pmatrix} 1 & \cdots & \cdots & 1 \\ a+b & \ddots & & \vdots \\ \vdots & \ddots & & \vdots \\ a+b & \cdots & a+b & 1 \end{pmatrix}$$

```
$A = \begin{pNiceMatrix}
1      & \Cdots &      & 1      \\\
a+b    &      &      & \Vdots \\\
\Vdots & \Ddots & \Ddots & \\\
a+b    & \Cdots & a+b  & 1
\end{pNiceMatrix}$
```

$$A = \begin{pmatrix} 1 & \cdots & \cdots & 1 \\ a+b & & & \vdots \\ \vdots & \ddots & & \vdots \\ a+b & \cdots & a+b & 1 \end{pmatrix}$$

It's possible to turn off the parallelization with the option `parallelize-diags` set to `false`:

The same example without parallelization:

$$A = \begin{pmatrix} 1 & \cdots & \cdots & 1 \\ a+b & \ddots & & \vdots \\ \vdots & \ddots & & \vdots \\ a+b & \cdots & a+b & 1 \end{pmatrix}$$

⁵⁰The command `\vrule` is a TeX (and not LaTeX) command.

⁵¹Of course, such rule is defined by the classical technics of `nicematrix` and, for this reason, won't cross the double rules of `\hline\hline`.

⁵²We speak of the lines created by `\Ddots` and not the lines created by a command `\line` in the `\CodeAfter`.

It's possible to specify the instruction `\Ddots` which will be drawn first (and which will be used to draw the other diagonal dotted lines when the parallelization is in force) with the key `draw-first:` `\Ddots[draw-first]`.

16.3 The “empty” cells

An instruction like `\Ldots`, `\Cdots`, etc. tries to determine the first non-empty cell on both sides. When the key `corners` is used (cf. p. 10), `nicematrix` computes corners consisting of empty cells. However, an “empty cell” is not necessarily a cell with no TeX content (that is to say a cell with no token between the two ampersands `&`). The precise rules are as follow.

- An implicit cell is empty. For example, in the following matrix:

```
\begin{pmatrix}
a & b & \\
c & & \\
\end{pmatrix}
```

the last cell (second row and second column) is empty.

- Each cell whose TeX output has a width equal to zero is empty.
- A cell containing the command `\NotEmpty` is not empty (and a PGF/Tikz node) is created in that cell.
- A cell with a command `\Hspace` (or `\Hspace*`) is empty. This command `\Hspace` is a command defined by the package `nicematrix` with the same meaning as `\hspace` except that the cell where it is used is considered as empty. This command can be used to fix the width of some columns of the matrix without interfering with `nicematrix`.
- A cell of a column of type `p`, `m` or `t` is always considered as not empty. *Caution* : One should not rely upon that point because it may change in a future version of `nicematrix`. On the other side, a cell of a column of type `V` of `varwidth` (cf. p. 20) is empty when its TeX content has a width equal to zero.

16.4 The option `exterior-arraycolsep`

The environment `{array}` inserts an horizontal space equal to `\arraycolsep` before and after each column. In particular, there is a space equal to `\arraycolsep` before and after the array. This feature of the environment `{array}` was probably not a good idea⁵³. The environment `{matrix}` of `amsmath` and its variants (`{pmatrix}`, `{vmatrix}`, etc.) of `amsmath` prefer to delete these spaces with explicit instructions `\hskip -\arraycolsep`⁵⁴. The package `nicematrix` does the same in all its environments, `{NiceArray}` included. However, if the user wants the environment `{NiceArray}` behaving by default like the environment `{array}` (for example, when adapting an existing document) it's possible to control this behaviour with the option `exterior-arraycolsep`, set by the command `\NiceMatrixOptions`. With this option, exterior spaces of length `\arraycolsep` will be inserted in the environments `{NiceArray}` (the other environments of `nicematrix` are not affected).

⁵³In the documentation of `{amsmath}`, we can read: *The extra space of `\arraycolsep` that `array` adds on each side is a waste so we remove it [in `{matrix}`] (perhaps we should instead remove it from `array` in general, but that's a harder task).*

⁵⁴And not by inserting `@{}` on both sides of the preamble of the array. As a consequence, the length of the `\hline` is not modified and may appear too long, in particular when using square brackets.

16.5 Incompatibilities

The package `nicematrix` is not fully compatible with the package `arydshln` (because this package redefines many internal of `array`).

Anyway, in order to use `arydshln`, one must first free the letter “:” by giving a new letter for the vertical dotted rules of `nicematrix`:

```
\NiceMatrixOptions{letter-for-dotted-lines=;}
```

The package `nicematrix` is not compatible with the class `ieeeaccess` (because that class is not compatible with PGF/Tikz).⁵⁵

In order to use `nicematrix` with the class `aastex631`, you have to add the following lines in the preamble of your document :

```
\BeforeBegin{NiceTabular}{\let\begin\BeginEnvironment\let\end\EndEnvironment}
\BeforeBegin{NiceArray}{\let\begin\BeginEnvironment}
\BeforeBegin{NiceMatrix}{\let\begin\BeginEnvironment}
```

In order to use `nicematrix` with the class `sn-jnl`, `pgf` must be loaded before the `\documentclass`:

```
\RequirePackage{pgf}
\documentclass{sn-jnl}
```

17 Examples

17.1 Utilisation of the key “tikz” of the command `\Block`

The key `tikz` of the command `\Block` is available only when Tikz is loaded.⁵⁶
For the following example, we need also the Tikz library `patterns`.

```
\usetikzlibrary{patterns}

\ttfamily \small
\begin{NiceTabular}{X[m]X[m]X[m]}[hvlines,cell-space-limits=3pt]
  \Block[tikz={pattern=grid,pattern color=lightgray}]{ }
    {pattern = grid,\ \ pattern color = lightgray}
& \Block[tikz={pattern = north west lines,pattern color=blue}]{ }
    {pattern = north west lines,\ \ pattern color = blue}
& \Block[tikz={outer color = red!50, inner color=white }]{2-1}
    {outer color = red!50,\ \ inner color = white} \ \
  \Block[tikz={pattern = sixpointed stars, pattern color = blue!15}]{ }
    {pattern = sixpointed stars,\ \ pattern color = blue!15}
& \Block[tikz={left color = blue!50}]{ }
    {left color = blue!50} \ \
\end{NiceTabular}
```

<pre>pattern = grid, pattern color = lightgray</pre>	<pre>pattern = north west lines, pattern color = blue</pre>	<pre>outer color = red!50, inner color = white</pre>
<pre>* pattern = sixpointed stars, * pattern color = blue!15 *</pre>	<pre>left color = blue!50</pre>	

⁵⁵See <https://tex.stackexchange.com/questions/528975/error-loading-tikz-in-ieeeaccess-class>

⁵⁶By default, `nicematrix` only loads PGF, which is a sub-layer of Tikz.

17.2 Notes in the tabulars

The tools provided by `nicematrix` for the composition of the tabular notes have been presented in the section 12 p. 32.

Let's consider that we wish to number the notes of a tabular with stars.⁵⁷

First, we write a command `\stars` similar the well-known commands `\arabic`, `\alph`, `\Alph`, etc. which produces a number of stars equal to its argument ⁵⁸

```
\ExplSyntaxOn
\NewDocumentCommand \stars { m }
{ \prg_replicate:nn { \value { #1 } } { $ \star $ } }
\ExplSyntaxOff
```

Of course, we change the style of the labels with the key `notes/style`. However, it would be interesting to change also some parameters in the type of list used to compose the notes at the end of the tabular. First, we required a composition flush right for the labels with the setting `align=right`. Moreover, we want the labels to be composed on a width equal to the width of the widest label. The widest label is, of course, the label with the greatest number of stars. We know that number: it is equal to `\value{tabularnote}` (because `tabularnote` is the LaTeX counter used by `\tabularnote` and, therefore, at the end of the tabular, its value is equal to the total number of tabular notes). We use the key `widest*` of `enumitem` in order to require a width equal to that value: `widest*=\value{tabularnote}`.

```
\NiceMatrixOptions
{
  notes =
  {
    style = \stars{#1} ,
    enumitem-keys =
    {
      widest* = \value{tabularnote} ,
      align = right
    }
  }
}

\begin{NiceTabular}{{}}llr{{}}
\toprule \RowStyle{\bfseries}
Last name & First name & Birth day \\
\midrule
Achard\tabularnote{Achard is an old family of the Poitou.}
& Jacques & 5 juin 1962 \\
Lefebvre\tabularnote{The name Lefebvre is an alteration of the name Lefebure.}
& Mathilde & 23 mai 1988 \\
Vanesse & Stephany & 30 octobre 1994 \\
Dupont & Chantal & 15 janvier 1998 \\
\bottomrule
\end{NiceTabular}
```

⁵⁷Of course, it's realistic only when there is very few notes in the tabular.

⁵⁸In fact: the value of its argument.

Last name	First name	Birth day
Achard*	Jacques	June 5, 2005
Lefebvre**	Mathilde	January 23, 1975
Vanesse	Stephany	October 30, 1994
Dupont	Chantal	January 15, 1998

*Achard is an old family of the Poitou.

**The name Lefebvre is an alteration of the name Lefebure.

17.3 Dotted lines

An example with the resultant of two polynoms:

```
\setlength{\extrarowheight}{1mm}
\[\begin{vNiceArray}{ccc}[columns-width=6mm]
a_0 & & & & b_0 & & & \\
a_1 & & \Ddots & & b_1 & & \Ddots & \\
\vdots & & \Ddots & & \vdots & & \Ddots & b_0 \\
a_p & & & a_0 & & & b_1 & \\
& & \Ddots & & a_1 & & \vdots & \\
& & & \vdots & & & \Ddots & \\
& & & a_p & & & b_q & \\
\end{vNiceArray}\]
```

$$\left(\begin{array}{ccccccc} a_0 & & & & & & \\ & \ddots & & & & & \\ & & \ddots & & & & \\ & & & \ddots & & & \\ & & & & \ddots & & \\ & & & & & \ddots & \\ & & & & & & \ddots \end{array} \right) \left(\begin{array}{ccccccc} b_0 & & & & & & \\ & \ddots & & & & & \\ & & \ddots & & & & \\ & & & \ddots & & & \\ & & & & \ddots & & \\ & & & & & \ddots & \\ & & & & & & \ddots \end{array} \right)$$

An example for a linear system:

```
\begin{pNiceArray}{*6c|c}[nullify-dots,last-col,code-for-last-col=\scriptstyle]
1 & 1 & 1 & \Cdots & 1 & 0 & \\
0 & 1 & 0 & \Cdots & 0 & & L_2 \gets L_2-L_1 \\
0 & 0 & 1 & \Ddots & \vdots & & L_3 \gets L_3-L_1 \\
& & & \Ddots & & \vdots & \\
\vdots & & & \Ddots & 0 & & \\
0 & & & \Cdots & 0 & 1 & L_n \gets L_n-L_1 \\
\end{pNiceArray}
```

$$\left(\begin{array}{ccccccc|c} 1 & 1 & 1 & \dots & 1 & & 0 \\ 0 & 1 & 0 & \dots & 0 & & \\ 0 & 0 & 1 & \ddots & \vdots & & \\ \vdots & & & \ddots & \vdots & & \\ \vdots & & & & \ddots & & \\ 0 & \dots & \dots & 0 & 1 & & \end{array} \right) \begin{array}{l} L_2 \leftarrow L_2 - L_1 \\ L_3 \leftarrow L_3 - L_1 \\ \vdots \\ L_n \leftarrow L_n - L_1 \end{array}$$

17.4 Dotted lines which are no longer dotted

The option `line-style` controls the style of the lines drawn by `\Ldots`, `\Cdots`, etc. Thus, it's possible with these commands to draw lines which are not longer dotted.

```

\NiceMatrixOptions{code-for-first-row = \scriptstyle,code-for-first-col = \scriptstyle }
\setcounter{MaxMatrixCols}{12}
\newcommand{\blue}{\color{blue}}
\[ \begin{pNiceMatrix}[last-row,last-col,nullify-dots,xdots/line-style={dashed,blue}]
1& & \Vdots & & & \Vdots \\
& \Ddots[line-style=standard] \\
& & 1 \\
& \Cdots[color=blue,line-style=dashed]& & & \blue 0 & \\
& \Cdots & & & \blue 1 & & & \Cdots & \blue \leftarrow i \\
& & & & 1 \\
& & & \Vdots & & \Ddots[line-style=standard] & & \Vdots \\
& & & & & & & 1 \\
& \Cdots & & & \blue 1 & \Cdots & & \Cdots & \blue 0 & & \Cdots & \blue \leftarrow j \\
& & & & & & & & & & 1 \\
& & & & & & & \Ddots[line-style=standard] \\
& & & \Vdots & & & & \Vdots & & & 1 \\
& & & \blue \overset{\uparrow}{i} & & & & \blue \overset{\uparrow}{j} \\
\end{pNiceMatrix} \]

```

$$\left(\begin{array}{cc|cc} 1 & \dots & & \\ & & 1 & \\ \hline & 0 & & 1 \\ & & 1 & \dots & 1 \\ \hline & 1 & & 0 \\ & & & & 1 & \dots & 1 \\ \hline \end{array} \right) \begin{array}{l} \\ \\ \leftarrow i \\ \\ \leftarrow j \\ \end{array}$$

In fact, it's even possible to draw solid lines with the commands `\Cdots`, `\Vdots`, etc.⁵⁹

```

\NiceMatrixOptions
  {nullify-dots,code-for-first-col = \color{blue},code-for-first-row=\color{blue}}
$\begin{pNiceMatrix}[first-row,first-col]
    & & \Ldots[line-style={solid,<->},shorten=0pt]^{n \text{ columns}} \\
    & 1 & 1 & 1 & \Ldots & 1 \\
    & 1 & 1 & 1 & & 1 \\
\VDots[line-style={solid,<->}]_n^{n \text{ rows}} & 1 & 1 & 1 & & 1 \\
    & 1 & 1 & 1 & & 1 \\
    & 1 & 1 & 1 & \Ldots & 1
\end{pNiceMatrix}$

```

$$\begin{matrix} & \xleftrightarrow{n \text{ columns}} \\ \begin{matrix} \uparrow \\ n \text{ rows} \\ \downarrow \end{matrix} & \begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & 1 & 1 & & 1 \\ 1 & 1 & 1 & & 1 \\ 1 & 1 & 1 & & 1 \\ 1 & 1 & 1 & \dots & 1 \end{pmatrix} \end{matrix}$$

⁵⁹In this document, the Tikz library `arrows.meta` has been loaded, which impacts the shape of the arrow tips.

17.5 Stacks of matrices

We often need to compose mathematical matrices on top on each other (for example for the resolution of linear systems).

In order to have the columns aligned one above the other, it's possible to fix a width for all the columns. That's what is done in the following example with the environment `{NiceMatrixBlock}` and its option `auto-columns-width`.

```
\begin{NiceMatrixBlock}[auto-columns-width]
\NiceMatrixOptions
{
  light-syntax,
  last-col, code-for-last-col = \color{blue} \scriptstyle,
}
\setlength{\extrarowheight}{1mm}

$\begin{pNiceArray}{rrrr|r}
12 -8 7 5 3 {} ;
3 -18 12 1 4 ;
-3 -46 29 -2 -15 ;
9 10 -5 4 7
\end{pNiceArray}$

\smallskip
$\begin{pNiceArray}{rrrr|r}
12 -8 7 5 3 ;
0 64 -41 1 19 { L_2 \gets L_1-4L_2 } ;
0 -192 123 -3 -57 { L_3 \gets L_1+4L_3 } ;
0 -64 41 -1 -19 { L_4 \gets 3L_1-4L_4 } ;
\end{pNiceArray}$

\smallskip
$\begin{pNiceArray}{rrrr|r}
12 -8 7 5 3 ;
0 64 -41 1 19 ;
0 0 0 0 0 { L_3 \gets 3 L_2 + L_3 }
\end{pNiceArray}$

\smallskip
$\begin{pNiceArray}{rrrr|r}
12 -8 7 5 3 {} ;
0 64 -41 1 19 ;
\end{pNiceArray}$

\end{NiceMatrixBlock}
```

$$\left(\begin{array}{cccc|c} 12 & -8 & 7 & 5 & 3 \\ 3 & -18 & 12 & 1 & 4 \\ -3 & -46 & 29 & -2 & -15 \\ 9 & 10 & -5 & 4 & 7 \end{array} \right)$$

$$\left(\begin{array}{cccc|c} 12 & -8 & 7 & 5 & 3 \\ 0 & 64 & -41 & 1 & 19 \\ 0 & -192 & 123 & -3 & -57 \\ 0 & -64 & 41 & -1 & -19 \end{array} \right) \begin{array}{l} L_2 \leftarrow L_1 - 4L_2 \\ L_3 \leftarrow L_1 + 4L_3 \\ L_4 \leftarrow 3L_1 - 4L_4 \end{array}$$

$$\left(\begin{array}{cccc|c} 12 & -8 & 7 & 5 & 3 \\ 0 & 64 & -41 & 1 & 19 \\ 0 & 0 & 0 & 0 & 0 \end{array}\right)_{L_3 \leftarrow 3L_2 + L_3}$$

$$\left(\begin{array}{cccc|c} 12 & -8 & 7 & 5 & 3 \\ 0 & 64 & -41 & 1 & 19 \end{array}\right)$$

However, one can see that the last matrix is not perfectly aligned with others. That's why, in LaTeX, the parenthesis have not exactly the same width (smaller parenthesis are a bit slimer).

In order to solve that problem, it's possible to require the delimiters to be composed with the maximal width, thanks to the boolean key `delimiters/max-width`.

```
\begin{NiceMatrixBlock}[auto-columns-width]
\NiceMatrixOptions
{
  delimiters/max-width,
  light-syntax,
  last-col, code-for-last-col = \color{blue}\scriptstyle,
}
\setlength{\extrarowheight}{1mm}

$\begin{pNiceArray}{rrrr|r}
12 -8 7 5 3 {} ;
3 -18 12 1 4 ;
-3 -46 29 -2 -15 ;
9 10 -5 4 7
\end{pNiceArray}$

...
\end{NiceMatrixBlock}
```

$$\left(\begin{array}{cccc|c} 12 & -8 & 7 & 5 & 3 \\ 3 & -18 & 12 & 1 & 4 \\ -3 & -46 & 29 & -2 & -15 \\ 9 & 10 & -5 & 4 & 7 \end{array}\right)$$

$$\left(\begin{array}{cccc|c} 12 & -8 & 7 & 5 & 3 \\ 0 & 64 & -41 & 1 & 19 \\ 0 & -192 & 123 & -3 & -57 \\ 0 & -64 & 41 & -1 & -19 \end{array}\right)_{\begin{array}{l} L_2 \leftarrow L_1 - 4L_2 \\ L_3 \leftarrow L_1 + 4L_3 \\ L_4 \leftarrow 3L_1 - 4L_4 \end{array}}$$

$$\left(\begin{array}{cccc|c} 12 & -8 & 7 & 5 & 3 \\ 0 & 64 & -41 & 1 & 19 \\ 0 & 0 & 0 & 0 & 0 \end{array}\right)_{L_3 \leftarrow 3L_2 + L_3}$$

$$\left(\begin{array}{cccc|c} 12 & -8 & 7 & 5 & 3 \\ 0 & 64 & -41 & 1 & 19 \end{array}\right)$$

If you wish an alignment of the different matrices without the same width for all the columns, you can construct a unique array and place the parenthesis with commands `\SubMatrix` in the `\CodeAfter`. Of course, that array can't be broken by a page break.

```
\setlength{\extrarowheight}{1mm}
\[\begin{NiceMatrix}[ r, last-col=6, code-for-last-col = \scriptstyle \color{blue} ]
12 & -8 & & 7 & 5 & 3 \\
3 & -18 & & 12 & 1 & 4 \\
-3 & -46 & & 29 & -2 & -15 \end{NiceMatrix}]
```

```

9 & 10 & -5 & 4 & 7 \\[1mm]
12 & -8 & 7 & 5 & 3 \\
0 & 64 & -41 & 1 & 19 & L_2 \gets L_1-4L_2 \\
0 & -192 & 123 & -3 & -57 & L_3 \gets L_1+4L_3 \\
0 & -64 & 41 & -1 & -19 & L_4 \gets 3L_1-4L_4 \\[1mm]
12 & -8 & 7 & 5 & 3 \\
0 & 64 & -41 & 1 & 19 \\
0 & 0 & 0 & 0 & 0 & L_3 \gets 3L_2+L_3 \\[1mm]
12 & -8 & 7 & 5 & 3 \\
0 & 64 & -41 & 1 & 19 \\
\CodeAfter [sub-matrix/vlines=4]
\SubMatrix({1-1}{4-5})
\SubMatrix({5-1}{8-5})
\SubMatrix({9-1}{11-5})
\SubMatrix({12-1}{13-5})
\end{NiceMatrix}\]

```

$$\begin{pmatrix} 12 & -8 & 7 & 5 & 3 \\ 3 & -18 & 12 & 1 & 4 \\ -3 & -46 & 29 & -2 & -15 \\ 9 & 10 & -5 & 4 & 7 \end{pmatrix}$$

$$\begin{pmatrix} 12 & -8 & 7 & 5 & 3 \\ 0 & 64 & -41 & 1 & 19 \\ 0 & -192 & 123 & -3 & -57 \\ 0 & -64 & 41 & -1 & -19 \end{pmatrix}
\begin{array}{l} L_2 \leftarrow L_1 - 4L_2 \\ L_3 \leftarrow L_1 + 4L_3 \\ L_4 \leftarrow 3L_1 - 4L_4 \end{array}$$

$$\begin{pmatrix} 12 & -8 & 7 & 5 & 3 \\ 0 & 64 & -41 & 1 & 19 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}
\begin{array}{l} L_3 \leftarrow 3L_2 + L_3 \end{array}$$

$$\begin{pmatrix} 12 & -8 & 7 & 5 & 3 \\ 0 & 64 & -41 & 1 & 19 \end{pmatrix}$$

In this tabular, the instructions `\SubMatrix` are executed after the composition of the tabular and, thus, the vertical rules are drawn without adding space between the columns.

In fact, it's possible, with the key `vlines-in-sub-matrix`, to choose a letter in the preamble of the array to specify vertical rules which will be drawn in the `\SubMatrix` only (by adding space between the columns).

```

\setlength{\extrarowheight}{1mm}
\[\begin{NiceArray}
[
  vlines-in-sub-matrix=I,
  last-col,
  code-for-last-col = \scriptstyle \color{blue}
]
{rrrrIr}
12 & -8 & 7 & 5 & 3 \\
3 & -18 & 12 & 1 & 4 \\
-3 & -46 & 29 & -2 & -15 \\
9 & 10 & -5 & 4 & 7 \\[1mm]
12 & -8 & 7 & 5 & 3 \\
0 & 64 & -41 & 1 & 19 & L_2 \gets L_1-4L_2 \\
0 & -192 & 123 & -3 & -57 & L_3 \gets L_1+4L_3
\end{NiceArray}\]

```

```

0 & -64 & 41 & -1 & -19 & L_4 \gets 3L_1-4L_4 \\[1mm]
12 & -8 & 7 & 5 & 3 \\
0 & 64 & -41 & 1 & 19 \\
0 & 0 & 0 & 0 & 0 & L_3 \gets 3L_2+L_3 \\[1mm]
12 & -8 & 7 & 5 & 3 \\
0 & 64 & -41 & 1 & 19 \\
\CodeAfter
\SubMatrix({1-1}{4-5})
\SubMatrix({5-1}{8-5})
\SubMatrix({9-1}{11-5})
\SubMatrix({12-1}{13-5})
\end{NiceArray}\]

```

$$\begin{pmatrix} 12 & -8 & 7 & 5 & 3 \\ 3 & -18 & 12 & 1 & 4 \\ -3 & -46 & 29 & -2 & -15 \\ 9 & 10 & -5 & 4 & 7 \end{pmatrix}
\begin{pmatrix} 12 & -8 & 7 & 5 & 3 \\ 0 & 64 & -41 & 1 & 19 \\ 0 & -192 & 123 & -3 & -57 \\ 0 & -64 & 41 & -1 & -19 \end{pmatrix}
\begin{matrix} \\ L_2 \leftarrow L_1 - 4L_2 \\ L_3 \leftarrow L_1 + 4L_3 \\ L_4 \leftarrow 3L_1 - 4L_4 \end{matrix}
\begin{pmatrix} 12 & -8 & 7 & 5 & 3 \\ 0 & 64 & -41 & 1 & 19 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 64 & -41 & 1 & 19 \end{pmatrix}
\begin{matrix} \\ \\ L_3 \leftarrow 3L_2 + L_3 \\ \end{matrix}$$

17.6 How to highlight cells of a matrix

In order to highlight a cell of a matrix, it's possible to “draw” that cell with the key `draw` of the command `\Block` (this is one of the uses of a mono-cell block⁶⁰).

```

$\begin{pNiceArray}{>\strut}cccc[margin,rules/color=blue]
\Block[draw]{a_{11}} & a_{12} & a_{13} & a_{14} \\
a_{21} & \Block[draw]{a_{22}} & a_{23} & a_{24} \\
a_{31} & a_{32} & \Block[draw]{a_{33}} & a_{34} \\
a_{41} & a_{42} & a_{43} & \Block[draw]{a_{44}} \\
\end{pNiceArray}$

```

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix}$$

We should remark that the rules we have drawn are drawn *after* the construction of the array and thus, they don't spread the cells of the array. We recall that, on the other side, the commands `\hline` and `\Hline`, the specifier “|” and the options `hlines`, `vlines`, `hvlines` and `hvlines-except-borders` spread the cells.⁶¹

⁶⁰We recall that, if the first mandatory argument of the command `\Block` is left empty, that means that the block is a mono-cell block

⁶¹For the command `\cline`, see the remark p. 8.

It's possible to color a row with `\rowcolor` in the `code-before` (or with `\rowcolor` in the first cell of the row if the key `colortbl-like` is used—even when `colortbl` is not loaded).

```
\begin{pNiceArray}{>{\strut}cccc}[margin, extra-margin=2pt,colortbl-like]
  \rowcolor{red!15}A_{11} & A_{12} & A_{13} & A_{14} \\
  A_{21} & \rowcolor{red!15}A_{22} & A_{23} & A_{24} \\
  A_{31} & A_{32} & \rowcolor{red!15}A_{33} & A_{34} \\
  A_{41} & A_{42} & A_{43} & \rowcolor{red!15}A_{44}
\end{pNiceArray}
```

$$\begin{pmatrix} A_{11} & A_{12} & A_{13} & A_{14} \\ A_{21} & A_{22} & A_{23} & A_{24} \\ A_{31} & A_{32} & A_{33} & A_{34} \\ A_{41} & A_{42} & A_{43} & A_{44} \end{pmatrix}$$

However, it's not possible to do a fine tuning. That's why we describe now a method to highlight a row of the matrix.

That example and the following ones require Tikz (by default, `nicematrix` only loads PGF, which is a sub-layer of Tikz) and the Tikz library `fit`. The following lines in the preamble of your document do the job:

```
\usepackage{tikz}
\usetikzlibrary{fit}
```

We create a rectangular Tikz node which encompasses the nodes of the second row by using the tools of the Tikz library `fit`. Those nodes are not available by default in the `\CodeBefore` (for efficiency). We have to require their creation with the key `create-cell-nodes` of the keyword `\CodeBefore`.

```
\tikzset{highlight/.style={rectangle,
                           fill=red!15,
                           rounded corners = 0.5 mm,
                           inner sep=1pt,
                           fit=#1}}

$\begin{bNiceMatrix}
\CodeBefore [create-cell-nodes]
  \tikz \node [highlight = (2-1) (2-3)] {} ;
\Body
0 & \Cdots & 0 \\
1 & \Cdots & 1 \\
0 & \Cdots & 0 \\
\end{bNiceMatrix}$
```

$$\begin{bmatrix} 0 & \cdots & 0 \\ 1 & \cdots & 1 \\ 0 & \cdots & 0 \end{bmatrix}$$

We consider now the following matrix. If we want to highlight each row of this matrix, we can use the previous technique three times.

```
\[\begin{pNiceArray}{ccc}[last-col]
\CodeBefore [create-cell-nodes]
  \begin{tikzpicture}
    \node [highlight = (1-1) (1-3)] {} ;
    \node [highlight = (2-1) (2-3)] {} ;
    \node [highlight = (3-1) (3-3)] {} ;
```

```

\end{tikzpicture}
\Body
a & a + b & a + b + c & L_1 \\
a & a      & a + b      & L_2 \\
a & a      & a          & L_3
\end{pNiceArray}\]

```

$$\begin{pmatrix} a & a+b & a+b+c \\ a & a & a+b \\ a & a & a \end{pmatrix} \begin{matrix} L_1 \\ L_2 \\ L_3 \end{matrix}$$

The result may seem disappointing. We can improve it by using the “medium nodes” instead of the “normal nodes”.

```

\[\begin{pNiceArray}{ccc}[last-col,create-medium-nodes]
\CodeBefore [create-cell-nodes]
\begin{tikzpicture} [name suffix = -medium]
\node [highlight = (1-1) (1-3)] {} ;
\node [highlight = (2-1) (2-3)] {} ;
\node [highlight = (3-1) (3-3)] {} ;
\end{tikzpicture}
\Body
a & a + b & a + b + c & L_1 \\
a & a      & a + b      & L_2 \\
a & a      & a          & L_3
\end{pNiceArray}\]

```

$$\begin{pmatrix} a & a+b & a+b+c \\ a & a & a+b \\ a & a & a \end{pmatrix} \begin{matrix} L_1 \\ L_2 \\ L_3 \end{matrix}$$

17.7 Utilisation of \SubMatrix in the \CodeBefore

In the following example, we illustrate the mathematical product of two matrices.

The whole figure is an environment `\NiceArray` and the three pairs of parenthesis have been added with `\SubMatrix` in the `\CodeBefore`.

$$L_i \begin{pmatrix} a_{11} & \dots & a_{1n} \\ \vdots & & \vdots \\ a_{i1} & \dots & a_{in} \\ \vdots & & \vdots \\ a_{n1} & \dots & a_{nn} \end{pmatrix} \begin{pmatrix} b_{11} & \dots & b_{1n} \\ \vdots & & \vdots \\ b_{kj} & & \vdots \\ \vdots & & \vdots \\ b_{n1} & \dots & b_{nn} \end{pmatrix}$$

```

\tikzset{highlight/.style={rectangle,
fill=red!15,
rounded corners = 0.5 mm,
inner sep=1pt,
fit=#1}}

```



```

\[\begin{NiceArray}{*{6}{c}@{\hspace{6mm}}*{5}{c}}[nullify-dots]
\CodeBefore [create-cell-nodes]
  \SubMatrix({2-7}{6-11})
  \SubMatrix({7-2}{11-6})
  \SubMatrix({7-7}{11-11})
  \begin{tikzpicture}
    \node [highlight = (9-2) (9-6)] { } ;
    \node [highlight = (2-9) (6-9)] { } ;
  \end{tikzpicture}
\Body
  & & & & & & & \color{blue}\scriptstyle C_j \\
  & & & & & & b_{11} & \Cdots & b_{1j} & \Cdots & b_{1n} \\
  & & & & & & \Vdots & & \Vdots & & \Vdots \\
  & & & & & & & & b_{kj} \\
  & & & & & & & & \Vdots \\
  & & & & & & b_{n1} & \Cdots & b_{nj} & \Cdots & b_{nn} \\
  & a_{11} & \Cdots & & a_{1n} \\
  & \Vdots & & & \Vdots & & \Vdots \\
\color{blue}\scriptstyle L_i
  & a_{i1} & \Cdots & a_{ik} & \Cdots & a_{in} & \Cdots & & c_{ij} \\
  & \Vdots & & & & \Vdots \\
  & a_{n1} & \Cdots & & a_{nn} \\
\CodeAfter
\tikz \draw [gray,shorten > = 1mm, shorten < = 1mm] (9-4.north) to [bend left] (4-9.west) ;
\end{NiceArray}\]

```

18 Implementation

By default, the package `nicematrix` doesn't patch any existing code.

However, when the option `renew-dots` is used, the commands `\cdots`, `\ldots`, `\dots`, `\vdots`, `\ddots` and `\iddots` are redefined in the environments provided by `nicematrix` as explained previously. In the same way, if the option `renew-matrix` is used, the environment `{matrix}` of `amsmath` is redefined.

On the other hand, the environment `{array}` is never redefined.

Of course, the package `nicematrix` uses the features of the package `array`. It tries to be independent of its implementation. Unfortunately, it was not possible to be strictly independent. For example, the package `nicematrix` relies upon the fact that the package `{array}` uses `\ialign` to begin the `\halign`.

Declaration of the package and packages loaded

The prefix `nicematrix` has been registered for this package.

See: <http://mirrors.ctan.org/macros/latex/contrib/l3kernel/l3prefixes.pdf>

<@@=nicematrix>

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

```

1 \RequirePackage{pgfcore}
2 \usepgfmodule{shapes}

```

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

```

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

```

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

We load some packages. The package `xparse` is still loaded for use on Overleaf. However, since oct. 2021, Overleaf uses TeXLive 2021 and we will be able to delete that row.

```

9 \RequirePackage { xparse }
10 \RequirePackage { array }
11 \RequirePackage { amsmath }

12 \cs_new_protected:Npn \@@_error:n { \msg_error:nn { nicematrix } }
13 \cs_new_protected:Npn \@@_error:nn { \msg_error:nnn { nicematrix } }
14 \cs_new_protected:Npn \@@_error:nnn { \msg_error:nnnn { nicematrix } }
15 \cs_new_protected:Npn \@@_fatal:n { \msg_fatal:nn { nicematrix } }
16 \cs_new_protected:Npn \@@_fatal:nn { \msg_fatal:nnn { nicematrix } }
17 \cs_new_protected:Npn \@@_msg_new:nn { \msg_new:nnn { nicematrix } }
18 \cs_new_protected:Npn \@@_msg_new:nnn { \msg_new:nnnn { nicematrix } }

19 \cs_new_protected:Npn \@@_msg_redirect_name:nn
20   { \msg_redirect_name:nnn { nicematrix } }

```

Technical definitions

```

21 \bool_new:N \c_@@_in_preamble_bool
22 \bool_set_true:N \c_@@_in_preamble_bool
23 \hook_gput_code:nnn { begindocument } { . }
24   { \bool_set_false:N \c_@@_in_preamble_bool }

25 \bool_new:N \c_@@_arydshln_loaded_bool
26 \bool_new:N \c_@@_booktabs_loaded_bool
27 \bool_new:N \c_@@_enumitem_loaded_bool
28 \bool_new:N \c_@@_tabularx_loaded_bool
29 \bool_new:N \c_@@_tikz_loaded_bool
30 \bool_new:N \c_@@_varwidth_loaded_bool
31 \hook_gput_code:nnn { begindocument } { . }
32   {
33     \@ifpackageloaded { varwidth }
34       { \bool_set_true:N \c_@@_varwidth_loaded_bool }
35       { }
36     \@ifpackageloaded { arydshln }
37       { \bool_set_true:N \c_@@_arydshln_loaded_bool }
38       { }
39     \@ifpackageloaded { booktabs }
40       { \bool_set_true:N \c_@@_booktabs_loaded_bool }
41       { }
42     \@ifpackageloaded { enumitem }
43       { \bool_set_true:N \c_@@_enumitem_loaded_bool }
44       { }
45     \@ifpackageloaded { tabularx }
46       { \bool_set_true:N \c_@@_tabularx_loaded_bool }
47       { }
48     \@ifpackageloaded { tikz }
49     {

```

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

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

```

50     \bool_set_true:N \c_@@_tikz_loaded_bool
51     \tl_const:Nn \c_@@_pgfortikzpicture_tl { \exp_not:N \tikzpicture }
52     \tl_const:Nn \c_@@_endpgfortikzpicture_tl { \exp_not:N \endtikzpicture }
53   }
54   {
55     \tl_const:Nn \c_@@_pgfortikzpicture_tl { \exp_not:N \pgfpicture }
56     \tl_const:Nn \c_@@_endpgfortikzpicture_tl { \exp_not:N \endpgfpicture }
57   }
58 }

```

We test whether the current class is revtex4-1 (deprecated) or revtex4-2 because these classes redefines `\array` (of `array`) in a way incompatible with our programming. At the date January 2021, the current version revtex4-2 is 4.2e (compatible with booktabs).

```

59 \bool_new:N \c_@@_revtex_bool
60 \ifclassloaded { revtex4-1 }
61   { \bool_set_true:N \c_@@_revtex_bool }
62   { }
63 \ifclassloaded { revtex4-2 }
64   { \bool_set_true:N \c_@@_revtex_bool }
65   { }

```

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

```

66 \cs_if_exist:NT \rvtx@ifformat@geq { \bool_set_true:N \c_@@_revtex_bool }

67 \cs_generate_variant:Nn \tl_if_single_token_p:n { V }

```

The following regex will be used to modify the preamble of the array when the key `colortbl-like` is used.

```

68 \regex_const:Nn \c_@@_columncolor_regex { \c { columncolor } }

```

If the final user uses `nicematrix`, PGF/Tikz will write instruction `\pgfsyspdfmark` in the aux file. If he changes its mind and no longer loads `nicematrix`, an error may occur at the next compilation because of remanent instructions `\pgfsyspdfmark` in the aux file. With the following code, we try to avoid that situation.

```

69 \cs_new_protected:Npn \@@_provide_pgfsyspdfmark:
70   {
71     \iow_now:Nn \@mainaux
72     {
73       \ExplSyntaxOn
74       \cs_if_free:NT \pgfsyspdfmark
75         { \cs_set_eq:NN \pgfsyspdfmark \@gobblethree }
76       \ExplSyntaxOff
77     }
78     \cs_gset_eq:NN \@@_provide_pgfsyspdfmark: \prg_do_nothing:
79   }

```

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

```

80 \ProvideDocumentCommand \iddots { }
81   {
82     \mathinner
83     {
84       \tex_mkern:D 1 mu
85       \box_move_up:nn { 1 pt } { \hbox:n { . } }
86       \tex_mkern:D 2 mu
87       \box_move_up:nn { 4 pt } { \hbox:n { . } }
88       \tex_mkern:D 2 mu
89       \box_move_up:nn { 7 pt }
90       { \vbox:n { \kern 7 pt \hbox:n { . } } }
91       \tex_mkern:D 1 mu
92     }

```

```
93 }
```

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.

```
94 \hook_gput_code:nnn { begindocument } { . }
95 {
96   \@ifpackageloaded { booktabs }
97     { \iow_now:Nn \@mainaux \nicematrix@redefine@check@rerun }
98     { }
99 }
100 \cs_set_protected:Npn \nicematrix@redefine@check@rerun
101 {
102   \cs_set_eq:NN \@_old_pgful@check@rerun \pgfutil@check@rerun
```

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

```
103   \cs_set_protected:Npn \pgfutil@check@rerun ##1 ##2
104   {
105     \str_if_eq:eeF { nm- } { \tl_range:nnn { ##1 } 1 3 }
106     { \@_old_pgful@check@rerun { ##1 } { ##2 } }
107   }
108 }
```

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

```
109 \bool_new:N \c_@@_colortbl_loaded_bool
110 \hook_gput_code:nnn { begindocument } { . }
111 {
112   \@ifpackageloaded { colortbl }
113     { \bool_set_true:N \c_@@_colortbl_loaded_bool }
114     { }
```

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.

```
115   \cs_set_protected:Npn \CT@arc@ { }
116   \cs_set:Npn \arrayrulecolor #1 # { \CT@arc@ { #1 } }
117   \cs_set:Npn \CT@arc@ #1 #2
118   {
119     \dim_compare:nNnT \baselineskip = \c_zero_dim \noalign
120       { \cs_gset:Npn \CT@arc@ { \color #1 { #2 } } }
121   }
```

Idem for `\CT@drs@`.

```
122   \cs_set:Npn \doublerulesepcolor #1 # { \CT@drs@ { #1 } }
123   \cs_set:Npn \CT@drs@ #1 #2
124   {
125     \dim_compare:nNnT \baselineskip = \c_zero_dim \noalign
126       { \cs_gset:Npn \CT@drsc@ { \color #1 { #2 } } }
127   }
128   \cs_set:Npn \hline
129   {
130     \noalign { \ifnum 0 = ` } \fi
131     \cs_set_eq:NN \hskip \vskip
132     \cs_set_eq:NN \vrule \hrule
133     \cs_set_eq:NN \@width \@height
134     { \CT@arc@ \vline }
135     \futurelet \reserved@a
136     \@xhline
137   }
138 }
139 }
```

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

```

140 \cs_set:Npn \@@_standard_cline #1 { \@@_standard_cline:w #1 \q_stop }
141 \cs_set:Npn \@@_standard_cline:w #1-#2 \q_stop
142 {
143   \int_compare:nNnT \l_@@_first_col_int = 0 { \omit & }
144   \int_compare:nNnT { #1 } > 1 { \multispan { \@@_pred:n { #1 } } & }
145   \multispan { \int_eval:n { #2 - #1 + 1 } }
146   {
147     \CT@arc@
148     \leaders \hrule \@height \arrayrulewidth \hfill

```

The following `\skip_horizontal:N \c_zero_dim` is to prevent a potential `\unskip` to delete the `\leaders`⁶²

```

149   \skip_horizontal:N \c_zero_dim
150 }

```

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

```

151 \everycr { }
152 \cr
153 \noalign { \skip_vertical:N -\arrayrulewidth }
154 }

```

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

```

155 \cs_set:Npn \@@_cline

```

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

```

156 { \@@_cline_i:en \l_@@_first_col_int }

```

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

```

157 \cs_set:Npn \@@_cline_i:nn #1 #2 { \@@_cline_i:w #1|#2- \q_stop }
158 \cs_set:Npn \@@_cline_i:w #1|#2-#3 \q_stop
159 {
160   \tl_if_empty:nTF { #3 }
161     { \@@_cline_iii:w #1|#2-#2 \q_stop }
162     { \@@_cline_ii:w #1|#2-#3 \q_stop }
163   }
164 \cs_set:Npn \@@_cline_ii:w #1|#2-#3-\q_stop
165 { \@@_cline_iii:w #1|#2-#3 \q_stop }
166 \cs_set:Npn \@@_cline_iii:w #1|#2-#3 \q_stop
167 {

```

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).

```

168   \int_compare:nNnT { #1 } < { #2 }
169     { \multispan { \int_eval:n { #2 - #1 } } & }
170   \multispan { \int_eval:n { #3 - #2 + 1 } }
171   {
172     \CT@arc@
173     \leaders \hrule \@height \arrayrulewidth \hfill
174     \skip_horizontal:N \c_zero_dim
175   }

```

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

```

176   \peek_meaning_remove_ignore_spaces:NTF \cline
177     { & \@@_cline_i:en { \@@_succ:n { #3 } } }
178     { \everycr { } \cr }

```

⁶²See question 99041 on TeX StackExchange.

```

179 }
180 \cs_generate_variant:Nn \@@_cline_i:nn { e n }

```

The following commands are only for efficiency. They must *not* be protected because it will be used (for instance) in names of PGF nodes.

```

181 \cs_new:Npn \@@_succ:n #1 { \the \numexpr #1 + 1 \relax }
182 \cs_new:Npn \@@_pred:n #1 { \the \numexpr #1 - 1 \relax }

```

The following command is a small shortcut.

```

183 \cs_new:Npn \@@_math_toggle_token:
184 { \bool_if:NF \l_@@_NiceTabular_bool \c_math_toggle_token }

```

```

185 \cs_new_protected:Npn \@@_set_CT@arc@:
186 { \peek_meaning:NTF [ \@@_set_CT@arc@_i: \@@_set_CT@arc@_ii: }
187 \cs_new_protected:Npn \@@_set_CT@arc@_i: [ #1 ] #2 \q_stop
188 { \cs_set:Npn \CT@arc@ { \color [ #1 ] { #2 } } }
189 \cs_new_protected:Npn \@@_set_CT@arc@_ii: #1 \q_stop
190 { \cs_set:Npn \CT@arc@ { \color { #1 } } }

```

```

191 \cs_new_protected:Npn \@@_set_CT@drsc@:
192 { \peek_meaning:NTF [ \@@_set_CT@drsc@_i: \@@_set_CT@drsc@_ii: }
193 \cs_new_protected:Npn \@@_set_CT@drsc@_i: [ #1 ] #2 \q_stop
194 { \cs_set:Npn \CT@drsc@ { \color [ #1 ] { #2 } } }
195 \cs_new_protected:Npn \@@_set_CT@drsc@_ii: #1 \q_stop
196 { \cs_set:Npn \CT@drsc@ { \color { #1 } } }

```

```

197 \cs_set_eq:NN \@@_old_pgfpointanchor \pgfpointanchor

```

The column S of siunitx

We want to know whether the package siunitx is loaded and, if it is loaded, we redefine the S columns of siunitx.

```

198 \bool_new:N \c_@@_siunitx_loaded_bool
199 \hook_gput_code:nnn { begindocument } { . }
200 {
201   \@ifpackageloaded { siunitx }
202   { \bool_set_true:N \c_@@_siunitx_loaded_bool }
203   { }
204 }

```

The command \@@_renew_NC@rewrite@S: will be used in each environment of nicematrix in order to “rewrite” the S column in each environment.

```

205 \hook_gput_code:nnn { begindocument } { . }
206 {
207   \bool_if:nTF { ! \c_@@_siunitx_loaded_bool }
208   { \cs_set_eq:NN \@@_renew_NC@rewrite@S: \prg_do_nothing: }
209   {
210     \cs_new_protected:Npn \@@_renew_NC@rewrite@S:
211     {
212       \renewcommand*{\NC@rewrite@S}[1] []
213       {

```

\@temptokena is a toks (not supported by the L3 programming layer).

```

214       \@temptokena \exp_after:wN
215       { \tex_the:D \@temptokena \@@_S: [ ##1 ] }
216       \NC@find
217     }
218   }
219 }
220 }

```

Parameters

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

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

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

```
222 \cs_new:Npn \@@_env: { nm - \int_use:N \g_@@_env_int }
```

The command `\NiceMatrixLastEnv` is not used by the package `nicematrix`. It's only a facility given to the final user. It gives the number of the last environment (in fact the number of the current environment but it's meant to be used after the environment in order to refer to that environment — and its nodes — without having to give it a name). This command *must* be expandable since it will be used in pgf nodes.

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

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

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

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

```
227 \int_new:N \g_@@_NiceMatrixBlock_int
```

The dimension `\l_@@_columns_width_dim` will be used when the options specify that all the columns must have the same width (but, if the key `columns-width` is used with the special value `auto`, the boolean `\l_@@_auto_columns_width_bool` also will be raised).

```
228 \dim_new:N \l_@@_columns_width_dim
```

The dimension `\l_@@_col_width_dim` will be available in each cell which belongs to a column of fixed width: `w{...}{...}`, `W{...}{...}`, `p{}`, `m{}`, `b{}` but also `X` (when the actual width of that column is known, that is to say after the first compilation). It's the width of that column. It will be used by some commands `\Block`. A non positive value means that the column has no fixed width (it's a column of type `c`, `r`, `l`, etc.).

```
229 \dim_new:N \l_@@_col_width_dim
230 \dim_set:Nn \l_@@_col_width_dim { -1 cm }
```

The following counters will be used to count the numbers of rows and columns of the array.

```
231 \int_new:N \g_@@_row_total_int
232 \int_new:N \g_@@_col_total_int
```

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

```
233 \int_new:N \l_@@_key_nb_rows_int
```

The following token list will contain the type of horizontal alignment of the current cell as provided by the corresponding column. The possible values are `r`, `l`, `c`. For exemple, a column `p[1]{3cm}` will provide the value `l` for all the cells of the column.

```
234 \str_new:N \l_@@_hpos_cell_str
235 \str_set:Nn \l_@@_hpos_cell_str { c }
```

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

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

Idem pour the mono-row blocks.

```
237 \dim_new:N \g_@@_blocks_ht_dim
238 \dim_new:N \g_@@_blocks_dp_dim
```

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

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

The sequence `\g_@@_names_seq` will be the list of all the names of environments used (via the option `name`) in the document: two environments must not have the same name. However, it's possible to use the option `allow-duplicate-names`.

```
240 \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.

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

If the user uses `{NiceArray}` or `{NiceTabular}` the flag `\l_@@_NiceArray_bool` will be raised.

```
242 \bool_new:N \l_@@_NiceArray_bool
```

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

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

```
243 \bool_new:N \l_@@_NiceTabular_bool
```

If the user uses `{NiceTabular*}`, the width of the tabular (in the first argument of the environment `{NiceTabular*}`) will be stored in the following dimension.

```
244 \dim_new:N \l_@@_tabular_width_dim
```

If the user uses an environment without preamble, we will raise the following flag.

```
245 \bool_new:N \l_@@_Matrix_bool
```

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

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

In a cell, it will be possible to know whether we are in a cell of a column of type `X` thanks to that flag.

```
247 \bool_new:N \l_@@_X_column_bool
```

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

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

```
249 \cs_new_protected:Npn \@@_test_if_math_mode:
250 {
251   \if_mode_math: \else:
252     \@@_fatal:n { Outside~math~mode }
253   \fi:
254 }
```

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

```
255 \tl_new:N \l_@@_letter_vlism_tl
```


The list of the columns where vertical lines in sub-matrices (vlism) must be drawn. Of course, the actual value of this sequence will be known after the analyse of the preamble of the array.

```
256 \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”.

```
257 \colorlet { nicematrix-last-col } { . }
258 \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`).

```
259 \str_new:N \g_@@_name_env_str
```

The following string will contain the word *command* or *environment* whether we are in a command of `nicematrix` or in an environment of `nicematrix`. The default value is *environment*.

```
260 \tl_set:Nn \g_@@_com_or_env_str { environment }
```

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

```
261 \cs_new:Npn \@@_full_name_env:
262 {
263   \str_if_eq:VnTF \g_@@_com_or_env_str { command }
264     { command \space \c_backslash_str \g_@@_name_env_str }
265     { environment \space \{ \g_@@_name_env_str \} }
266 }
```

The following token list corresponds to the option `code-after` (it’s also possible to set the value of that parameter with the keyword `\CodeAfter`). That parameter is *public*.

```
267 \tl_new:N \g_nicematrix_code_after_tl
```

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

```
268 \tl_new:N \l_@@_code_tl
```

The following token list has a function similar to `\g_nicematrix_code_after_tl` but it is used internally by `nicematrix`. In fact, we have to distinguish between `\g_nicematrix_code_after_tl` and `\g_@@_internal_code_after_tl` because we must take care of the order in which instructions stored in that parameters are executed.

```
269 \tl_new:N \g_@@_internal_code_after_tl
```

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.

```
270 \int_new:N \l_@@_old_iRow_int
271 \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 token list corresponds to the key `rules/color` available in the environments.

```
272 \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 optional argument between square brackets. The default value, of course, is 1.

```
273 \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.

```
274 \bool_new:N \l_@@_X_columns_aux_bool
275 \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.

```
276 \bool_new:N \g_@@_after_col_zero_bool
```

A kind of false row will be inserted at the end of the array for the construction of the **col** nodes (and also to fix the width of the columns when `columns-width` is used). When this special row will be created, we will raise the flag `\g_@@_row_of_col_done_bool` in order to avoid some actions set in the redefinition of `\everycr` when the last `\cr` of the `\halign` will occur (after that row of **col** nodes).

```
277 \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).

```
278 \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`).

```
279 \tl_new:N \l_@@_code_before_tl
280 \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).

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

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

```
282 \dim_new:N \l_@@_x_initial_dim
283 \dim_new:N \l_@@_y_initial_dim
284 \dim_new:N \l_@@_x_final_dim
285 \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.

```
286 \dim_zero_new:N \l_@@_tmpc_dim
287 \dim_zero_new:N \l_@@_tmpd_dim
```

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

```
288 \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”.

```
289 \dim_new:N \g_@@_width_last_col_dim
290 \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.

```
291 \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!).

```
292 \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}`.

```
293 \seq_new:N \g_@@_pos_of_xdots_seq
```

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

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

```
294 \seq_new:N \g_@@_pos_of_stroken_blocks_seq
```

If the user has used the key `corners` (or the key `hvlines-except-corners`, even though that key is deprecated), all the cells which are in an (empty) corner will be stored in the following sequence.

```
295 \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`).

```
296 \seq_new:N \g_@@_submatrix_names_seq
```

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

```
297 \bool_new:N \l_@@_width_used_bool
```

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

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

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

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

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

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

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

```
303 \int_new:N \l_@@_col_max_int
```

The following sequence will be used when the command `\SubMatrix` is used in the `\CodeBefore` (and not in the `\CodeAfter`). It will contain the position of all the sub-matrices specified in the `code-before`. Each sub-matrix is represented by an “object” of the forme $\{i\}\{j\}\{k\}\{l\}$ 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.

```
304 \seq_new:N \g_@@_submatrix_seq
```

We are able to determine the number of columns specified in the preamble (for the environments with explicit preamble of course and without the potential exterior columns).

```
305 \int_new:N \g_@@_static_num_of_col_int
```

The following parameters correspond to the keys `fill`, `draw`, `tikz`, `borders`, and `rounded-corners` of the command `\Block`.

```
306 \tl_new:N \l_@@_fill_tl
307 \tl_new:N \l_@@_draw_tl
308 \seq_new:N \l_@@_tikz_seq
309 \clist_new:N \l_@@_borders_clist
310 \dim_new:N \l_@@_rounded_corners_dim
```

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

The following token list correspond to the key `color` of the command `\Block`.

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

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

```
312 \dim_new:N \l_@@_line_width_dim
```

The parameters of the horizontal position of the label of a block. If the user uses the key `c` or `C`, the value is `c`. If the user uses the key `l` or `L`, the value is `l`. If the user uses the key `r` or `R`, the value is `r`. If the user has used a capital letter, the boolean `\l_@@_hpos_of_block_cap_bool` will be raised (in the second pass of the analyze of the keys of the command `\Block`).

```
313 \str_new:N \l_@@_hpos_block_str
314 \str_set:Nn \l_@@_hpos_block_str { c }
315 \bool_new:N \l_@@_hpos_of_block_cap_bool
```

For the vertical position, the possible values are `c`, `t` and `b`. Of course, it would be interesting to program a key `T` and a key `B`.

```
316 \tl_new:N \l_@@_vpos_of_block_tl
317 \tl_set:Nn \l_@@_vpos_of_block_tl { c }
```

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

```
318 \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).

```
319 \bool_new:N \l_@@_vlines_block_bool
320 \bool_new:N \l_@@_hlines_block_bool
```

The blocks which use the key `-` will store their content in a box. These boxes are numbered with the following counter.

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

322 \dim_new:N \l_@@_submatrix_extra_height_dim
323 \dim_new:N \l_@@_submatrix_left_xshift_dim
324 \dim_new:N \l_@@_submatrix_right_xshift_dim
325 \clist_new:N \l_@@_hlines_clist
326 \clist_new:N \l_@@_vlines_clist
327 \clist_new:N \l_@@_submatrix_hlines_clist
328 \clist_new:N \l_@@_submatrix_vlines_clist
```

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

```
329 \bool_new:N \l_@@_dotted_bool
```

Variables for the exterior rows and columns

The keys for the exterior rows and columns are `first-row`, `first-col`, `last-row` and `last-col`. However, internally, these keys are not coded in a similar way.

• First row

The integer `\l_@@_first_row_int` is the number of the first row of the array. The default value is 1, but, if the option `first-row` is used, the value will be 0.

```
330 \int_new:N \l_@@_first_row_int
331 \int_set:Nn \l_@@_first_row_int 1
```

• First column

The integer `\l_@@_first_col_int` is the number of the first column of the array. The default value is 1, but, if the option `first-col` is used, the value will be 0.

```
332 \int_new:N \l_@@_first_col_int
333 \int_set:Nn \l_@@_first_col_int 1
```

• Last row

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

```
334 \int_new:N \l_@@_last_row_int
335 \int_set:Nn \l_@@_last_row_int { -2 }
```

If, in an environment like `{pNiceArray}`, the option `last-row` is used without value, we will globally raise the following flag. It will be used to know if we have, after the construction of the array, to write in the `aux` file the number of the “last row”.⁶³

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

Idem for `\l_@@_last_col_without_value_bool`

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

• Last column

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

```
338 \int_new:N \l_@@_last_col_int
339 \int_set:Nn \l_@@_last_col_int { -2 }
```

⁶³We can’t use `\l_@@_last_row_int` for this usage because, if `nicematrix` has read its value from the `aux` file, the value of the counter won’t be `-1` any longer.

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

```
\begin{pNiceArray}{cc}[last-col]
1 & 2 \\
3 & 4
\end{pNiceArray}
```

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

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

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

Some utilities

```
341 \cs_set_protected:Npn \@@_cut_on_hyphen:w #1-#2\q_stop
342 {
343   \tl_set:Nn \l_tmpa_tl { #1 }
344   \tl_set:Nn \l_tmpb_tl { #2 }
345 }
```

The following takes as argument the name of a `clist` and which should be a list of intervals of integers. It *expands* that list, that is to say, it replaces (by a sort of `mapcan` or `flat_map`) the interval by the explicit list of the integers.

```
346 \cs_new_protected:Npn \@@_expand_clist:N #1
347 {
348   \clist_if_in:NnF #1 { all }
349   {
350     \clist_clear:N \l_tmpa_clist
351     \clist_map_inline:Nn #1
352     {
353       \tl_if_in:nnTF { ##1 } { - }
354       { \@@_cut_on_hyphen:w ##1 \q_stop }
355       {
356         \tl_set:Nn \l_tmpa_tl { ##1 }
357         \tl_set:Nn \l_tmpb_tl { ##1 }
358       }
359       \int_step_inline:nnn { \l_tmpa_tl } { \l_tmpb_tl }
360       { \clist_put_right:Nn \l_tmpa_clist { ####1 } }
361     }
362     \tl_set_eq:NN #1 \l_tmpa_clist
363   }
364 }
```

The command `\tablarnote`

The LaTeX counter `tablarnote` 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.

```
365 \newcounter { tablarnote }
```

We will store in the following sequence the tabular notes of a given array.

```
366 \seq_new:N \g_@@_tablarnotes_seq
```

However, before the actual tabular notes, it’s possible to put a text specified by the key `tablarnote` of the environment. The token list `\l_@@_tablarnote_tl` corresponds to the value of that key.

```
367 \tl_new:N \l_@@_tablarnote_tl
```

The following counter will be used to count the number of successive tabular notes such as in `\tabularnote{Note 1}\tabularnote{Note 2}\tabularnote{Note 3}`. In the tabular, the labels of those nodes are composed as a comma separated list (e.g. a,b,c).

```
368 \int_new:N \l_@@_number_of_notes_int
```

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

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

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

```
370 \cs_new:Npn \@@_notes_label_in_tabular:n #1 { \textsuperscript { #1 } }
```

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

```
371 \cs_new:Npn \@@_notes_label_in_list:n #1 { \textsuperscript { #1 } }
```

We define `\thetabularnote` because it will be used by LaTeX if the user want to reference a footnote which has been marked by a `\label`. The TeX group is for the case where the user has put an instruction such as `\color{red}` in `\@@_notes_style:n`.

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

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

```
373 \hook_gput_code:nnn { begindocument } { . }
374 {
375   \bool_if:nTF { ! \c_@@_enumitem_loaded_bool }
376   {
377     \NewDocumentCommand \tabularnote { m }
378     { \@@_error:n { enumitem-not-loaded } }
379   }
380 }
```

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

```
381 \newlist { tabularnotes } { enumerate } { 1 }
382 \setlist [ tabularnotes ]
383 {
384   topsep = 0pt ,
385   noitemsep ,
386   leftmargin = * ,
387   align = left ,
388   labelsep = 0pt ,
389   label =
390     \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotesi } } ,
391 }
392 \newlist { tabularnotes* } { enumerate* } { 1 }
393 \setlist [ tabularnotes* ]
394 {
395   afterlabel = \nobreak ,
396   itemjoin = \quad ,
397   label =
398     \@@_notes_label_in_list:n { \@@_notes_style:n { tabularnotes*i } }
399 }
```

The command `\tabularnote` is available in the whole document (and not only in the environments of `nicematrix`) because we want it to be available in the caption of a `{table}` (before the following `{NiceTabular}` or `{NiceArray}`). That's also the reason why the variables `\c@tabularnote` and `\g_@@_tabularnotes_seq` will be cleared at the end of the environment of `nicematrix` (and not at the beginning).

Unfortunately, if the package `caption` is loaded, the command `\caption` evaluates its argument twice and since it is not aware (of course) of `\tabularnote`, the command `\tabularnote` is, in fact, not usable in `\caption` when `caption` is loaded.⁶⁴

```

400     \NewDocumentCommand \tabularnote { m }
401     {
402         \bool_if:nTF { ! \l_@@_NiceArray_bool && \l_@@_in_env_bool }
403         { \@@_error:n { tabularnote~forbidden } }
404         {

```

`\l_@@_number_of_notes_int` is used to count the number of successive tabular notes such as in `\tabularnote{Note 1}\tabularnote{Note 2}\tabularnote{Note 3}`. We will have to compose the labels of these notes as a comma separated list (e.g. a,b,c).

```

405         \int_incr:N \l_@@_number_of_notes_int

```

We expand the content of the note at the point of use of `\tabularnote` as does `\footnote`.

```

406         \seq_gput_right:Nn \g_@@_tabularnotes_seq { #1 }
407         \peek_meaning:NF \tabularnote
408         {

```

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

```

409         \hbox_set:Nn \l_tmpa_box
410         {

```

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

```

411         \@@_notes_label_in_tabular:n
412         {
413             \stepcounter { tabularnote }
414             \@@_notes_style:n { tabularnote }
415             \prg_replicate:nn { \l_@@_number_of_notes_int - 1 }
416             {
417                 ,
418                 \stepcounter { tabularnote }
419                 \@@_notes_style:n { tabularnote }
420             }
421         }
422     }

```

We use `\refstepcounter` in order to have the (last) tabular note referenceable (with the standard command `\label`) and that's why we have to go back with a decrementation of the counter `tabularnote` first.

```

423         \addtocounter { tabularnote } { -1 }
424         \refstepcounter { tabularnote }
425         \int_zero:N \l_@@_number_of_notes_int
426         \hbox_overlap_right:n { \box_use:N \l_tmpa_box }

```

If the command `\tabularnote` is used exactly at the end of the cell, the `\unskip` (inserted by `array`?) will delete the skip we insert now and the label of the footnote will be composed in an overlapping position (by design).

```

427         \skip_horizontal:n { \box_wd:N \l_tmpa_box }
428     }
429 }
430 }
431 }
432 }

```

⁶⁴We should try to find a solution to that problem.

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.

```

433 \cs_new_protected:Npn \@@_pgf_rect_node:nnnnn #1 #2 #3 #4 #5
434 {
435   \begin { pgfscope }
436   \pgfset
437   {
438     outer~sep = \c_zero_dim ,
439     inner~sep = \c_zero_dim ,
440     minimum~size = \c_zero_dim
441   }
442   \pgftransformshift { \pgfpoint { 0.5 * ( #2 + #4 ) } { 0.5 * ( #3 + #5 ) } }
443   \pgfnode
444   { rectangle }
445   { center }
446   {
447     \vbox_to_ht:nn
448     { \dim_abs:n { #5 - #3 } }
449     {
450       \vfill
451       \hbox_to_wd:nn { \dim_abs:n { #4 - #2 } } { }
452     }
453   }
454   { #1 }
455   { }
456   \end { pgfscope }
457 }
```

The command `\@@_pgf_rect_node:nnn` is a variant of `\@@_pgf_rect_node:nnnnn`: it takes two PGF points as arguments instead of the four dimensions which are the coordinates.

```

458 \cs_new_protected:Npn \@@_pgf_rect_node:nnn #1 #2 #3
459 {
460   \begin { pgfscope }
461   \pgfset
462   {
463     outer~sep = \c_zero_dim ,
464     inner~sep = \c_zero_dim ,
465     minimum~size = \c_zero_dim
466   }
467   \pgftransformshift { \pgfpointscale { 0.5 } { \pgfpointadd { #2 } { #3 } } }
468   \pgfpointdiff { #3 } { #2 }
469   \pgfgetlastxy \l_tmpa_dim \l_tmpb_dim
470   \pgfnode
471   { rectangle }
472   { center }
473   {
474     \vbox_to_ht:nn
475     { \dim_abs:n \l_tmpb_dim }
476     { \vfill \hbox_to_wd:nn { \dim_abs:n \l_tmpa_dim } { } }
477   }
478   { #1 }
479   { }
480   \end { pgfscope }
481 }
```

The options

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 `colortbl-like` is used, these commands are available.

```
482 \bool_new:N \l_@@_colortbl_like_bool
```

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

```
483 \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`).

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

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.

```
486 \dim_new:N \l_@@_inter_dots_dim
487 \hook_gput_code:nnn { begindocument } { . }
488 { \dim_set:Nn \l_@@_inter_dots_dim { 0.45 em } }
```

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

The following dimension is the minimal distance between a node (in fact an anchor of that node) and a dotted line (we say “minimal” because, by definition, a dotted line is not a continuous line and, therefore, this distance may vary a little).

```
489 \dim_new:N \l_@@_xdots_shorten_dim
490 \hook_gput_code:nnn { begindocument } { . }
491 { \dim_set:Nn \l_@@_xdots_shorten_dim { 0.3 em } }
```

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

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.

```
492 \dim_new:N \l_@@_radius_dim
493 \hook_gput_code:nnn { begindocument } { . }
494 { \dim_set:Nn \l_@@_radius_dim { 0.53 pt } }
```

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

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.

```
495 \tl_new:N \l_@@_xdots_line_style_tl
496 \tl_const:Nn \c_@@_standard_tl { standard }
497 \tl_set_eq:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl
```

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

```
498 \bool_new:N \l_@@_light_syntax_bool
```

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

```
499 \tl_new:N \l_@@_baseline_tl
500 \tl_set:Nn \l_@@_baseline_tl c
```

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

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

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

```
502 \bool_new:N \l_@@_parallelize_diags_bool
503 \bool_set_true:N \l_@@_parallelize_diags_bool
```

The following parameter correspond to the key `corners`. The elements of that `clist` must be in NW, SW, NE and SE.

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

```
505 \dim_new:N \l_@@_notes_above_space_dim
506 \hook_gput_code:nnn { begindocument } { . }
507 { \dim_set:Nn \l_@@_notes_above_space_dim { 1 mm } }
```

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

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

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

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

```
509 \bool_new:N \l_@@_respect_arraystretch_bool
```

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

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

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

```
511 \bool_new:N \g_@@_recreate_cell_nodes_bool
```

The string `\l_@@_name_str` will contain the optional name of the environment: this name can be used to access to the Tikz nodes created in the array from outside the environment.

```
512 \str_new:N \l_@@_name_str
```

The boolean `\l_@@_medium_nodes_bool` will be used to indicate whether the “medium nodes” are created in the array. Idem for the “large nodes”.

```
513 \bool_new:N \l_@@_medium_nodes_bool
514 \bool_new:N \l_@@_large_nodes_bool
```

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

```
515 \bool_new:N \l_@@_except_borders_bool
```

The dimension `\l_@@_left_margin_dim` correspond to the option `left-margin`. Idem for the right margin. These parameters are involved in the creation of the “medium nodes” but also in the placement of the delimiters and the drawing of the horizontal dotted lines (`\hdottedline`).

```
516 \dim_new:N \l_@@_left_margin_dim
517 \dim_new:N \l_@@_right_margin_dim
```

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

```
518 \dim_new:N \l_@@_extra_left_margin_dim
519 \dim_new:N \l_@@_extra_right_margin_dim
```

The token list `\l_@@_end_of_row_tl` corresponds to the option `end-of-row`. It specifies the symbol used to mark the ends of rows when the light syntax is used.

```
520 \tl_new:N \l_@@_end_of_row_tl
521 \tl_set:Nn \l_@@_end_of_row_tl { ; }
```

The following parameter is for the color the dotted lines drawn by `\Cdots`, `\Ldots`, `\Vdots`, `\Ddots`, `\Iddots` and `\Hdotsfor` but *not* the dotted lines drawn by `\hdottedline` and “:”.

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

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

```
523 \tl_new:N \l_@@_delimiters_color_tl
```

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

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

```
525 \keys_define:nn { NiceMatrix / xdots }
526 {
527   line-style .code:n =
528   {
529     \bool_lazy_or:nnTF
```

We can't use `\c_@@_tikz_loaded_bool` to test whether `tikz` is loaded because `\NiceMatrixOptions` may be used in the preamble of the document.

```
530   { \cs_if_exist_p:N \tikzpicture }
531   { \str_if_eq_p:nn { #1 } { standard } }
532   { \tl_set:Nn \l_@@_xdots_line_style_tl { #1 } }
533   { \@_error:n { bad-option-for-line-style } }
534   } ,
535   line-style .value_required:n = true ,
536   color .tl_set:N = \l_@@_xdots_color_tl ,
537   color .value_required:n = true ,
538   shorten .dim_set:N = \l_@@_xdots_shorten_dim ,
539   shorten .value_required:n = true ,
```

The options `down` and `up` are not documented for the final user because he should use the syntax with `^` and `_`.

```
540   down .tl_set:N = \l_@@_xdots_down_tl ,
541   up .tl_set:N = \l_@@_xdots_up_tl ,
```

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

```
542   draw-first .code:n = \prg_do_nothing: ,
543   unknown .code:n = \@_error:n { Unknown-key-for-xdots }
544 }
```

```

545 \keys_define:nn { NiceMatrix / rules }
546 {
547   color .tl_set:N = \l_@@_rules_color_tl ,
548   color .value_required:n = true ,
549   width .dim_set:N = \arrayrulewidth ,
550   width .value_required:n = true
551 }

```

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

```

552 \keys_define:nn { NiceMatrix / Global }
553 {
554   custom-line .code:n = \@@_custom_line:n { #1 } ,
555   delimiters .code:n = \keys_set:nn { NiceMatrix / delimiters } { #1 } ,
556   delimiters .value_required:n = true ,
557   rules .code:n = \keys_set:nn { NiceMatrix / rules } { #1 } ,
558   rules .value_required:n = true ,
559   standard-cline .bool_set:N = \l_@@_standard_cline_bool ,
560   standard-cline .default:n = true ,
561   cell-space-top-limit .dim_set:N = \l_@@_cell_space_top_limit_dim ,
562   cell-space-top-limit .value_required:n = true ,
563   cell-space-bottom-limit .dim_set:N = \l_@@_cell_space_bottom_limit_dim ,
564   cell-space-bottom-limit .value_required:n = true ,
565   cell-space-limits .meta:n =
566   {
567     cell-space-top-limit = #1 ,
568     cell-space-bottom-limit = #1 ,
569   } ,
570   cell-space-limits .value_required:n = true ,
571   xdots .code:n = \keys_set:nn { NiceMatrix / xdots } { #1 } ,
572   light-syntax .bool_set:N = \l_@@_light_syntax_bool ,
573   light-syntax .default:n = true ,
574   end-of-row .tl_set:N = \l_@@_end_of_row_tl ,
575   end-of-row .value_required:n = true ,
576   first-col .code:n = \int_zero:N \l_@@_first_col_int ,
577   first-row .code:n = \int_zero:N \l_@@_first_row_int ,
578   last-row .int_set:N = \l_@@_last_row_int ,
579   last-row .default:n = -1 ,
580   code-for-first-col .tl_set:N = \l_@@_code_for_first_col_tl ,
581   code-for-first-col .value_required:n = true ,
582   code-for-last-col .tl_set:N = \l_@@_code_for_last_col_tl ,
583   code-for-last-col .value_required:n = true ,
584   code-for-first-row .tl_set:N = \l_@@_code_for_first_row_tl ,
585   code-for-first-row .value_required:n = true ,
586   code-for-last-row .tl_set:N = \l_@@_code_for_last_row_tl ,
587   code-for-last-row .value_required:n = true ,
588   hlines .clist_set:N = \l_@@_hlines_clist ,
589   vlines .clist_set:N = \l_@@_vlines_clist ,
590   hlines .default:n = all ,
591   vlines .default:n = all ,
592   vlines-in-sub-matrix .code:n =
593   {
594     \tl_if_single_token:nTF { #1 }
595     { \tl_set:Nn \l_@@_letter_vlism_tl { #1 } }
596     { \@@_error:n { One~letter~allowed } }
597   } ,
598   vlines-in-sub-matrix .value_required:n = true ,
599   hvlines .code:n =
600   {
601     \clist_set:Nn \l_@@_vlines_clist { all }
602     \clist_set:Nn \l_@@_hlines_clist { all }
603   } ,
604   hvlines-except-borders .code:n =

```

```

605     {
606       \clist_set:Nn \l_@@_vlines_clist { all }
607       \clist_set:Nn \l_@@_hlines_clist { all }
608       \bool_set_true:N \l_@@_except_borders_bool
609     } ,
610     parallelize-diags .bool_set:N = \l_@@_parallelize_diags_bool ,

```

With the option `renew-dots`, the command `\cdots`, `\ldots`, `\vdots`, `\ddots`, etc. are redefined and behave like the commands `\Cdots`, `\Ldots`, `\Vdots`, `\Ddots`, etc.

```

611     renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
612     renew-dots .value_forbidden:n = true ,
613     nullify-dots .bool_set:N = \l_@@_nullify_dots_bool ,
614     create-medium-nodes .bool_set:N = \l_@@_medium_nodes_bool ,
615     create-large-nodes .bool_set:N = \l_@@_large_nodes_bool ,
616     create-extra-nodes .meta:n =
617       { create-medium-nodes , create-large-nodes } ,
618     left-margin .dim_set:N = \l_@@_left_margin_dim ,
619     left-margin .default:n = \arraycolsep ,
620     right-margin .dim_set:N = \l_@@_right_margin_dim ,
621     right-margin .default:n = \arraycolsep ,
622     margin .meta:n = { left-margin = #1 , right-margin = #1 } ,
623     margin .default:n = \arraycolsep ,
624     extra-left-margin .dim_set:N = \l_@@_extra_left_margin_dim ,
625     extra-right-margin .dim_set:N = \l_@@_extra_right_margin_dim ,
626     extra-margin .meta:n =
627       { extra-left-margin = #1 , extra-right-margin = #1 } ,
628     extra-margin .value_required:n = true ,
629     respect-arraystretch .bool_set:N = \l_@@_respect_arraystretch_bool ,
630     respect-arraystretch .default:n = true
631   }

```

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

```

632 \keys_define:nn { NiceMatrix / Env }
633 {

```

The key `hvlines-except-corners` is now deprecated (use `hvlines` and `corners` instead).

```

634   hvlines-except-corners .code:n =
635     {
636       \clist_set:Nn \l_@@_corners_clist { #1 }
637       \clist_set:Nn \l_@@_vlines_clist { all }
638       \clist_set:Nn \l_@@_hlines_clist { all }
639     } ,
640   hvlines-except-corners .default:n = { NW , SW , NE , SE } ,
641   corners .clist_set:N = \l_@@_corners_clist ,
642   corners .default:n = { NW , SW , NE , SE } ,
643   code-before .code:n =
644     {
645       \tl_if_empty:nF { #1 }
646       {
647         \tl_put_right:Nn \l_@@_code_before_tl { #1 }
648         \bool_set_true:N \l_@@_code_before_bool
649       }
650     } ,

```

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

```

651   c .code:n = \tl_set:Nn \l_@@_baseline_tl c ,
652   t .code:n = \tl_set:Nn \l_@@_baseline_tl t ,
653   b .code:n = \tl_set:Nn \l_@@_baseline_tl b ,
654   baseline .tl_set:N = \l_@@_baseline_tl ,
655   baseline .value_required:n = true ,

```

```

656 columns-width .code:n =
657   \tl_if_eq:nnTF { #1 } { auto }
658   { \bool_set_true:N \l_@@_auto_columns_width_bool }
659   { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,
660 columns-width .value_required:n = true ,
661 name .code:n =

```

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

```

662   \legacy_if:nF { measuring@ }
663   {
664     \str_set:Nn \l_tmpa_str { #1 }
665     \seq_if_in:NVTF \g_@@_names_seq \l_tmpa_str
666     { \@@_error:nn { Duplicate-name } { #1 } }
667     { \seq_gput_left:NV \g_@@_names_seq \l_tmpa_str }
668     \str_set_eq:NN \l_@@_name_str \l_tmpa_str
669   } ,
670 name .value_required:n = true ,
671 code-after .tl_gset:N = \g_nicematrix_code_after_tl ,
672 code-after .value_required:n = true ,
673 colortbl-like .code:n =
674   \bool_set_true:N \l_@@_colortbl_like_bool
675   \bool_set_true:N \l_@@_code_before_bool ,
676 colortbl-like .value_forbidden:n = true
677 }
678 \keys_define:nn { NiceMatrix / notes }
679 {
680   para .bool_set:N = \l_@@_notes_para_bool ,
681   para .default:n = true ,
682   code-before .tl_set:N = \l_@@_notes_code_before_tl ,
683   code-before .value_required:n = true ,
684   code-after .tl_set:N = \l_@@_notes_code_after_tl ,
685   code-after .value_required:n = true ,
686   bottomrule .bool_set:N = \l_@@_notes_bottomrule_bool ,
687   bottomrule .default:n = true ,
688   style .code:n = \cs_set:Nn \@@_notes_style:n { #1 } ,
689   style .value_required:n = true ,
690   label-in-tabular .code:n =
691     \cs_set:Nn \@@_notes_label_in_tabular:n { #1 } ,
692   label-in-tabular .value_required:n = true ,
693   label-in-list .code:n =
694     \cs_set:Nn \@@_notes_label_in_list:n { #1 } ,
695   label-in-list .value_required:n = true ,
696   enumitem-keys .code:n =
697     {
698       \bool_if:NTF \c_@@_in_preamble_bool
699       {
700         \hook_gput_code:nnn { begindocument } { . }
701         {
702           \bool_if:NT \c_@@_enumitem_loaded_bool
703           { \setlist* [ tabularnotes ] { #1 } }
704         }
705       }
706       {
707         \bool_if:NT \c_@@_enumitem_loaded_bool
708         { \setlist* [ tabularnotes ] { #1 } }
709       }
710     } ,
711   enumitem-keys .value_required:n = true ,
712   enumitem-keys-para .code:n =
713     {
714       \bool_if:NTF \c_@@_in_preamble_bool
715       {
716         \hook_gput_code:nnn { begindocument } { . }

```

```

717         {
718             \bool_if:NT \c_@@_enumitem_loaded_bool
719             { \setlist* [ tabularnotes* ] { #1 } }
720         }
721     }
722     {
723         \bool_if:NT \c_@@_enumitem_loaded_bool
724         { \setlist* [ tabularnotes* ] { #1 } }
725     }
726 },
727 enumitem-keys-para .value_required:n = true ,
728 unknown .code:n = \@@_error:n { Unknown-key-for-notes }
729 }
730 \keys_define:nn { NiceMatrix / delimiters }
731 {
732     max-width .bool_set:N = \l_@@_delimiters_max_width_bool ,
733     max-width .default:n = true ,
734     color .tl_set:N = \l_@@_delimiters_color_tl ,
735     color .value_required:n = true ,
736 }

```

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

```

737 \keys_define:nn { NiceMatrix }
738 {
739     NiceMatrixOptions .inherit:n =
740         { NiceMatrix / Global } ,
741     NiceMatrixOptions / xdots .inherit:n = NiceMatrix / xdots ,
742     NiceMatrixOptions / rules .inherit:n = NiceMatrix / rules ,
743     NiceMatrixOptions / notes .inherit:n = NiceMatrix / notes ,
744     NiceMatrixOptions / delimiters .inherit:n = NiceMatrix / delimiters ,
745     NiceMatrixOptions / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
746     SubMatrix / rules .inherit:n = NiceMatrix / rules ,
747     CodeAfter / xdots .inherit:n = NiceMatrix / xdots ,
748     NiceMatrix .inherit:n =
749         {
750             NiceMatrix / Global ,
751             NiceMatrix / Env ,
752         } ,
753     NiceMatrix / xdots .inherit:n = NiceMatrix / xdots ,
754     NiceMatrix / rules .inherit:n = NiceMatrix / rules ,
755     NiceMatrix / delimiters .inherit:n = NiceMatrix / delimiters ,
756     NiceTabular .inherit:n =
757         {
758             NiceMatrix / Global ,
759             NiceMatrix / Env
760         } ,
761     NiceTabular / xdots .inherit:n = NiceMatrix / xdots ,
762     NiceTabular / rules .inherit:n = NiceMatrix / rules ,
763     NiceTabular / delimiters .inherit:n = NiceMatrix / delimiters ,
764     NiceArray .inherit:n =
765         {
766             NiceMatrix / Global ,
767             NiceMatrix / Env ,
768         } ,
769     NiceArray / xdots .inherit:n = NiceMatrix / xdots ,
770     NiceArray / rules .inherit:n = NiceMatrix / rules ,
771     NiceArray / delimiters .inherit:n = NiceMatrix / delimiters ,
772     pNiceArray .inherit:n =
773         {
774             NiceMatrix / Global ,
775             NiceMatrix / Env ,

```



```

776     } ,
777     pNiceArray / xdots .inherit:n = NiceMatrix / xdots ,
778     pNiceArray / rules .inherit:n = NiceMatrix / rules ,
779     pNiceArray / delimiters .inherit:n = NiceMatrix / delimiters ,
780 }

```

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

```

781 \keys_define:nn { NiceMatrix / NiceMatrixOptions }
782 {
783     width .code:n = \dim_set:Nn \l_@@_width_dim { #1 } ,
784     width .value_required:n = true ,
785     last-col .code:n = \tl_if_empty:nF { #1 }
786         { \@@_error:n { last-col~non-empty~for~NiceMatrixOptions } }
787         \int_zero:N \l_@@_last_col_int ,
788     small .bool_set:N = \l_@@_small_bool ,
789     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.

```

790     renew-matrix .code:n = \@@_renew_matrix: ,
791     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}`.

```

792     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.

```

793     columns-width .code:n =
794         \tl_if_eq:nnTF { #1 } { auto }
795         { \@@_error:n { Option~auto~for~columns-width } }
796         { \dim_set:Nn \l_@@_columns_width_dim { #1 } } ,

```

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

```

797     allow-duplicate-names .code:n =
798         \@@_msg_redirect_name:nn { Duplicate-name } { none } ,
799     allow-duplicate-names .value_forbidden:n = true ,

```

By default, the specifier used in the preamble of the array (for example in `{pNiceArray}`) to draw a vertical dotted line between two columns is the colon “:”. However, it’s possible to change this letter with `letter-for-dotted-lines` and, by the way, the letter “:” will remain free for other packages (for example `arydshln`).

```

800     letter-for-dotted-lines .code:n =
801     {
802         \tl_if_single_token:nTF { #1 }
803         { \str_set:Nx \l_@@_letter_for_dotted_lines_str { #1 } }
804         { \@@_error:n { One-letter~allowed } }
805     } ,
806     letter-for-dotted-lines .value_required:n = true ,
807     notes .code:n = \keys_set:nn { NiceMatrix / notes } { #1 } ,
808     notes .value_required:n = true ,
809     sub-matrix .code:n =
810         \keys_set:nn { NiceMatrix / sub-matrix } { #1 } ,
811     sub-matrix .value_required:n = true ,
812     unknown .code:n = \@@_error:n { Unknown~key~for~NiceMatrixOptions }
813 }

```

```

814 \str_new:N \l_@@_letter_for_dotted_lines_str
815 \str_set_eq:NN \l_@@_letter_for_dotted_lines_str \c_colon_str

```

`\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.

```

816 \NewDocumentCommand \NiceMatrixOptions { m }
817 { \keys_set:nn { NiceMatrix / NiceMatrixOptions } { #1 } }

```

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

```

818 \keys_define:nn { NiceMatrix / NiceMatrix }
819 {
820   last-col .code:n = \tl_if_empty:nTF {#1}
821   {
822     \bool_set_true:N \l_@@_last_col_without_value_bool
823     \int_set:Nn \l_@@_last_col_int { -1 }
824   }
825   { \int_set:Nn \l_@@_last_col_int { #1 } } ,
826   l .code:n = \tl_set:Nn \l_@@_type_of_col_tl l ,
827   r .code:n = \tl_set:Nn \l_@@_type_of_col_tl r ,
828   small .bool_set:N = \l_@@_small_bool ,
829   small .value_forbidden:n = true ,
830   unknown .code:n = \@@_error:n { Unknown-key-for-NiceMatrix }
831 }

```

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

```

832 \keys_define:nn { NiceMatrix / NiceArray }
833 {

```

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.

```

834   small .bool_set:N = \l_@@_small_bool ,
835   small .value_forbidden:n = true ,
836   last-col .code:n = \tl_if_empty:nF { #1 }
837   { \@@_error:n { last-col-non-empty-for-NiceArray } }
838   \int_zero:N \l_@@_last_col_int ,
839   notes / para .bool_set:N = \l_@@_notes_para_bool ,
840   notes / para .default:n = true ,
841   notes / bottomrule .bool_set:N = \l_@@_notes_bottomrule_bool ,
842   notes / bottomrule .default:n = true ,
843   tabularnote .tl_set:N = \l_@@_tabularnote_tl ,
844   tabularnote .value_required:n = true ,
845   r .code:n = \@@_error:n { r-or-l-with-preamble } ,
846   l .code:n = \@@_error:n { r-or-l-with-preamble } ,
847   unknown .code:n = \@@_error:n { Unknown-key-for-NiceArray }
848 }
849 \keys_define:nn { NiceMatrix / pNiceArray }
850 {
851   first-col .code:n = \int_zero:N \l_@@_first_col_int ,
852   last-col .code:n = \tl_if_empty:nF {#1}
853   { \@@_error:n { last-col-non-empty-for-NiceArray } }
854   \int_zero:N \l_@@_last_col_int ,
855   first-row .code:n = \int_zero:N \l_@@_first_row_int ,
856   small .bool_set:N = \l_@@_small_bool ,
857   small .value_forbidden:n = true ,
858   r .code:n = \@@_error:n { r~or~l~with~preamble } ,
859   l .code:n = \@@_error:n { r~or~l~with~preamble } ,
860   unknown .code:n = \@@_error:n { Unknown-key-for-NiceMatrix }
861 }

```

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

```
862 \keys_define:nn { NiceMatrix / NiceTabular }
863 {
```

The dimension width will be used if at least a column of type X is used. If there is no column of type X, an error will be raised.

```
864   width .code:n = \dim_set:Nn \l_@@_width_dim { #1 }
865           \bool_set_true:N \l_@@_width_used_bool ,
866   width .value_required:n = true ,
867   notes / para .bool_set:N = \l_@@_notes_para_bool ,
868   notes / para .default:n = true ,
869   notes / bottomrule .bool_set:N = \l_@@_notes_bottomrule_bool ,
870   notes / bottomrule .default:n = true ,
871   tabularnote .tl_set:N = \l_@@_tabularnote_tl ,
872   tabularnote .value_required:n = true ,
873   last-col .code:n = \tl_if_empty:nF {#1}
874           { \@@_error:n { last-col~non-empty~for~NiceArray } }
875           \int_zero:N \l_@@_last_col_int ,
876   r .code:n = \@@_error:n { r~or~l~with~preamble } ,
877   l .code:n = \@@_error:n { r~or~l~with~preamble } ,
878   unknown .code:n = \@@_error:n { Unknown-key-for-NiceTabular }
879 }
```

Important code used by {NiceArrayWithDelims}

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

```
880 \cs_new_protected:Npn \@@_cell_begin:w
881 {
```

The token list \g_@@_post_action_cell_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 (that’s why it’s called a *post-action*).

```
882   \tl_gclear:N \g_@@_post_action_cell_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).

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

We increment \c@jCol, which is the counter of the columns.

```
884   \int_gincr:N \c@jCol
```

Now, we increment the counter of the rows. We don’t do this incrementation in the \everycr because some packages, like arydshln, create special rows in the \halign that we don’t want to take into account.

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

The content of the cell is composed in the box \l_@@_cell_box. The \hbox_set_end: corresponding to this \hbox_set:Nw will be in the \@@_cell_end: (and the potential \c_math_toggle_token also).

```
887   \hbox_set:Nw \l_@@_cell_box
888   \bool_if:NF \l_@@_NiceTabular_bool
889   {
890     \c_math_toggle_token
891     \bool_if:NT \l_@@_small_bool \scriptstyle
892   }
```

For unexplained reason, with XeTeX (and not with the other engines), the environments of `nicematrix` were all composed in black and do not take into account the color of the encompassing text. As a workaround, you peek the color in force at the beginning of the environment and we use it now (in each cell of the array).

```
893 \color { nicematrix }
894 \g_@@_row_style_tl
```

We will call *corners* of the matrix the cases which are at the intersection of the exterior rows and exterior columns (of course, the four corners doesn't always exist simultaneously).

The codes `\l_@@_code_for_first_row_tl` and `al` don't apply in the corners of the matrix.

```
895 \int_compare:nNnTF \c@iRow = 0
896 {
897   \int_compare:nNnT \c@jCol > 0
898   {
899     \l_@@_code_for_first_row_tl
900     \xglobal \colorlet { nicematrix-first-row } { . }
901   }
902 }
903 {
904   \int_compare:nNnT \c@iRow = \l_@@_last_row_int
905   {
906     \l_@@_code_for_last_row_tl
907     \xglobal \colorlet { nicematrix-last-row } { . }
908   }
909 }
910 }
```

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.

```
911 \cs_new_protected:Npn \@@_begin_of_row:
912 {
913   \int_gincr:N \c@iRow
914   \dim_gset_eq:NN \g_@@_dp_ante_last_row_dim \g_@@_dp_last_row_dim
915   \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }
916   \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
917   \pgfpicture
918   \pgfrememberpicturepositiononpagetrue
919   \pgfcoordinate
920   { \@@_env: - row - \int_use:N \c@iRow - base }
921   { \pgfpoint \c_zero_dim { 0.5 \arrayrulewidth } }
922   \str_if_empty:NF \l_@@_name_str
923   {
924     \pgfnodealias
925     { \l_@@_name_str - row - \int_use:N \c@iRow - base }
926     { \@@_env: - row - \int_use:N \c@iRow - base }
927   }
928   \endpgfpicture
929 }
```

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.

```
930 \cs_new_protected:Npn \@@_update_for_first_and_last_row:
931 {
932   \int_compare:nNnTF \c@iRow = 0
933   {
934     \dim_gset:Nn \g_@@_dp_row_zero_dim
935     { \dim_max:nn \g_@@_dp_row_zero_dim { \box_dp:N \l_@@_cell_box } }
```

```

936     \dim_gset:Nn \g_@@_ht_row_zero_dim
937     { \dim_max:nn \g_@@_ht_row_zero_dim { \box_ht:N \l_@@_cell_box } }
938   }
939   {
940     \int_compare:nNnT \c@iRow = 1
941     {
942       \dim_gset:Nn \g_@@_ht_row_one_dim
943       { \dim_max:nn \g_@@_ht_row_one_dim { \box_ht:N \l_@@_cell_box } }
944     }
945   }
946 }
947 \cs_new_protected:Npn \@@_rotate_cell_box:
948 {
949   \box_rotate:Nn \l_@@_cell_box { 90 }
950   \int_compare:nNnT \c@iRow = \l_@@_last_row_int
951   {
952     \vbox_set_top:Nn \l_@@_cell_box
953     {
954       \vbox_to_zero:n { }
955       \skip_vertical:n { - \box_ht:N \@arstrutbox + 0.8 ex }
956       \box_use:N \l_@@_cell_box
957     }
958   }
959   \bool_gset_false:N \g_@@_rotate_bool
960 }
961 \cs_new_protected:Npn \@@_adjust_size_box:
962 {
963   \dim_compare:nNnT \g_@@_blocks_wd_dim > \c_zero_dim
964   {
965     \box_set_wd:Nn \l_@@_cell_box
966     { \dim_max:nn { \box_wd:N \l_@@_cell_box } \g_@@_blocks_wd_dim }
967     \dim_gzero:N \g_@@_blocks_wd_dim
968   }
969   \dim_compare:nNnT \g_@@_blocks_dp_dim > \c_zero_dim
970   {
971     \box_set_dp:Nn \l_@@_cell_box
972     { \dim_max:nn { \box_dp:N \l_@@_cell_box } \g_@@_blocks_dp_dim }
973     \dim_gzero:N \g_@@_blocks_dp_dim
974   }
975   \dim_compare:nNnT \g_@@_blocks_ht_dim > \c_zero_dim
976   {
977     \box_set_ht:Nn \l_@@_cell_box
978     { \dim_max:nn { \box_ht:N \l_@@_cell_box } \g_@@_blocks_ht_dim }
979     \dim_gzero:N \g_@@_blocks_ht_dim
980   }
981 }
982 \cs_new_protected:Npn \@@_cell_end:
983 {
984   \@@_math_toggle_token:
985   \hbox_set_end:

```

The token list `\g_@@_post_action_cell_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.

```

986   \g_@@_post_action_cell_tl
987   \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
988   \@@_adjust_size_box:
989   \box_set_ht:Nn \l_@@_cell_box
990   { \box_ht:N \l_@@_cell_box + \l_@@_cell_space_top_limit_dim }
991   \box_set_dp:Nn \l_@@_cell_box
992   { \box_dp:N \l_@@_cell_box + \l_@@_cell_space_bottom_limit_dim }

```

We want to compute in `\g_@@_max_cell_width_dim` the width of the widest cell of the array (except the cells of the “first column” and the “last column”).

```

993 \dim_gset:Nn \g_@@_max_cell_width_dim
994 { \dim_max:nn \g_@@_max_cell_width_dim { \box_wd:N \l_@@_cell_box } }

```

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

```

995 \@@_update_for_first_and_last_row:

```

If the cell is empty, or may be considered as if, we must not create the PGF node, for two reasons:

- it’s a waste of time since such a node would be rather pointless;
- we test the existence of these nodes in order to determine whether a cell is empty when we search the extremities of a dotted line.

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

- 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`, a `\llap` 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.

```

996 \bool_if:NTF \g_@@_empty_cell_bool
997 { \box_use_drop:N \l_@@_cell_box }
998 {
999   \bool_lazy_or:nnTF
1000     \g_@@_not_empty_cell_bool
1001     { \dim_compare_p:nNn { \box_wd:N \l_@@_cell_box } > \c_zero_dim }
1002     \@@_node_for_cell:
1003     { \box_use_drop:N \l_@@_cell_box }
1004   }
1005   \int_gset:Nn \g_@@_col_total_int { \int_max:nn \g_@@_col_total_int \c@jCol }
1006   \bool_gset_false:N \g_@@_empty_cell_bool
1007   \bool_gset_false:N \g_@@_not_empty_cell_bool
1008 }

```

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

```

1009 \cs_new_protected:Npn \@@_node_for_cell:
1010 {
1011   \pgfpicture
1012   \pgfsetbaseline \c_zero_dim
1013   \pgfrememberpicturepositiononpagetrue
1014   \pgfset
1015   {
1016     inner~sep = \c_zero_dim ,
1017     minimum~width = \c_zero_dim
1018   }
1019   \pgfnode
1020   { rectangle }
1021   { base }
1022   { \box_use_drop:N \l_@@_cell_box }
1023   { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1024   { }
1025   \str_if_empty:NF \l_@@_name_str
1026   {
1027     \pgfnodealias
1028     { \l_@@_name_str - \int_use:N \c@iRow - \int_use:N \c@jCol }
1029     { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol }
1030   }
1031   \endpgfpicture
1032 }

```

As its name says, the following command is a patch for the command `\@@_node_for_cell:`. This patch will be appended on the left of `\@@_node_for_the_cell:` when the construction of the cell nodes (of the form $(i-j)$) in the `\CodeBefore` is required.

```

1033 \cs_new_protected:Npn \@@_patch_node_for_cell:n #1
1034 {
1035   \cs_new_protected:Npn \@@_patch_node_for_cell:
1036   {
1037     \hbox_set:Nn \l_@@_cell_box
1038     {
1039       \box_move_up:nn { \box_ht:N \l_@@_cell_box}
1040       \hbox_overlap_left:n
1041       {
1042         \pgfsys@markposition
1043         { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - NW }

```

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

```

1044       #1
1045     }
1046     \box_use:N \l_@@_cell_box
1047     \box_move_down:nn { \box_dp:N \l_@@_cell_box }
1048     \hbox_overlap_left:n
1049     {
1050       \pgfsys@markposition
1051       { \@@_env: - \int_use:N \c@iRow - \int_use:N \c@jCol - SE }
1052       #1
1053     }
1054   }
1055 }
1056 }

```

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

```

1057 \bool_lazy_or:nnTF \sys_if_engine_xetex_p: \sys_if_output_dvi_p:
1058 {
1059   \@@_patch_node_for_cell:n
1060   { \skip_horizontal:n { 0.5 \box_wd:N \l_@@_cell_box } }
1061 }
1062 { \@@_patch_node_for_cell:n { } }

```

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

For example, for the following matrix,

```

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

```

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

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

```

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

```

The first argument is a boolean which indicates whether you must put the instruction on the left or on the right on the list of instructions.

```

1063 \cs_new_protected:Npn \@@_instruction_of_type:nnn #1 #2 #3
1064 {
1065   \bool_if:nTF { #1 } \tl_gput_left:cx \tl_gput_right:cx
1066   { g_@@_ #2 _ lines _ tl }

```

```

1067     {
1068       \use:c { @@ _ draw _ #2 : nnn }
1069       { \int_use:N \c@iRow }
1070       { \int_use:N \c@jCol }
1071       { \exp_not:n { #3 } }
1072     }
1073   }
1074   \cs_new_protected:Npn \@@_array:
1075   {
1076     \bool_if:NTF \l_@@_NiceTabular_bool
1077     { \dim_set_eq:NN \col@sep \tabcolsep }
1078     { \dim_set_eq:NN \col@sep \arraycolsep }
1079     \dim_compare:nNnTF \l_@@_tabular_width_dim = \c_zero_dim
1080     { \cs_set_nopar:Npn \@halignto { } }
1081     { \cs_set_nopar:Npx \@halignto { to \dim_use:N \l_@@_tabular_width_dim } }

```

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

```

1082   \@tabarray
1083   [ \str_if_eq:VnTF \l_@@_baseline_tl c c t ]
1084   }

```

`\l_@@_baseline_tl` may have the value `t`, `c` or `b`. However, if the value is `b`, we compose the `\array` (of array) with the option `t` and the right translation will be done further. Remark that `\str_if_eq:VnTF` is fully expandable and you need something fully expandable here.

We keep in memory the standard version of `\ialign` because we will redefine `\ialign` in the environment `{NiceArrayWithDelims}` but restore the standard version for use in the cells of the array.

```

1085   \cs_set_eq:NN \@@_old_ialign: \ialign

```

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

```

1086   \cs_new_protected:Npn \@@_create_row_node:
1087   {

```

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

```

1088     \hbox
1089     {
1090       \bool_if:NT \l_@@_code_before_bool
1091       {
1092         \vtop
1093         {
1094           \skip_vertical:N 0.5\arrayrulewidth
1095           \pgfsys@markposition { \@@_env: - row - \@@_succ:n \c@iRow }
1096           \skip_vertical:N -0.5\arrayrulewidth
1097         }
1098       }
1099       \pgfpicture
1100       \pgfrememberpicturepositiononpagetrue
1101       \pgfcoordinate { \@@_env: - row - \@@_succ:n \c@iRow }
1102       { \pgfpoint \c_zero_dim { - 0.5 \arrayrulewidth } }
1103       \str_if_empty:NF \l_@@_name_str
1104       {
1105         \pgfnodealias
1106         { \l_@@_name_str - row - \@@_succ:n \c@iRow }
1107         { \@@_env: - row - \@@_succ:n \c@iRow }
1108       }
1109       \endpgfpicture
1110     }
1111   }

```

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

```

1112   \cs_new:Npn \@@_everycr: { \noalign { \@@_everycr_i: } }

```



```

1113 \cs_new_protected:Npn \@@_everycr_i:
1114 {
1115     \int_gzero:N \c@jCol
1116     \bool_gset_false:N \g_@@_after_col_zero_bool
1117     \bool_if:NF \g_@@_row_of_col_done_bool
1118     {
1119         \@@_create_row_node:

```

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

```

1120     \tl_if_empty:NF \l_@@_hlines_clist
1121     {
1122         \tl_if_eq:NnF \l_@@_hlines_clist { all }
1123         {
1124             \exp_args:NNx
1125             \clist_if_in:NnT
1126             \l_@@_hlines_clist
1127             { \@@_succ:n \c@iRow }
1128         }
1129     }

```

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.

```

1130     \int_compare:nNnT \c@iRow > { -1 }
1131     {
1132         \int_compare:nNnF \c@iRow = \l_@@_last_row_int

```

The command `\CT@arc@` is a command of `colortbl` which sets the color of the rules in the array. The package `nicematrix` uses it even if `colortbl` is not loaded. We use a TeX group in order to limit the scope of `\CT@arc@`.

```

1133         { \hrule height \arrayrulewidth width \c_zero_dim }
1134     }
1135 }
1136 }
1137 }
1138 }

```

The command `\@@_newcolumntype` is the command `\newcolumntype` of `array` without the warnings for redefinitions of columns types (we will use it to redefine the columns types `w` and `W`).

```

1139 \cs_set_protected:Npn \@@_newcolumntype #1
1140 {
1141     \cs_set:cpn { NC @ find @ #1 } ##1 #1 { \NC@ { ##1 } }
1142     \peek_meaning:NTF [
1143         { \newcol@ #1 }
1144         { \newcol@ #1 [ 0 ] }
1145     }

```

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

```

1146 \cs_set_protected:Npn \@@_renew_dots:
1147 {
1148     \cs_set_eq:NN \ldots \@@_Ldots
1149     \cs_set_eq:NN \cdots \@@_Cdots
1150     \cs_set_eq:NN \vdots \@@_Vdots
1151     \cs_set_eq:NN \ddots \@@_Ddots
1152     \cs_set_eq:NN \iddots \@@_Iddots
1153     \cs_set_eq:NN \dots \@@_Ldots
1154     \cs_set_eq:NN \hdotsfor \@@_Hdotsfor:
1155 }

```

When the key `colortbl-like` is used, the following code will be executed.

```

1156 \cs_new_protected:Npn \@@_colortbl_like:
1157 {

```

```

1158 \cs_set_eq:NN \cellcolor \@@_cellcolor_tabular
1159 \cs_set_eq:NN \rowcolor \@@_rowcolor_tabular
1160 \cs_set_eq:NN \columncolor \@@_columncolor_preamble
1161 }

```

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

```

1162 \cs_new_protected:Npn \@@_pre_array_ii:
1163 {

```

For unexplained reason, with XeTeX (and not with the other engines), the environments of `nicematrix` were all composed in black and do not take into account the color of the encompassing text. As a workaround, you peek the color in force at the beginning of the environment and we will it in each cell.

```

1164 \xglobal \colorlet { nicematrix } { . }

```

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

```

1165 \int_gzero:N \g_@@_total_X_weight_int
1166 \@@_expand_clist:N \l_@@_hlines_clist
1167 \@@_expand_clist:N \l_@@_vlines_clist

```

If `booktabs` is loaded, we have to patch the macro `\@BTnormal` which is a macro of `booktabs`. The macro `\@BTnormal` draws an horizontal rule but it occurs after a vertical skip done by a low level TeX command. When this macro `\@BTnormal` occurs, the `row` node has yet been inserted by `nicematrix` *before* the vertical skip (and thus, at a wrong place). That why we decide to create a new `row` node (for the same row). We patch the macro `\@BTnormal` to create this `row` node. This new `row` node will overwrite the previous definition of that `row` node and we have managed to avoid the error messages of that redefinition ⁶⁵.

```

1168 \bool_if:NT \c_@@_booktabs_loaded_bool
1169 { \tl_put_left:Nn \@BTnormal \@@_create_row_node: }
1170 \box_clear_new:N \l_@@_cell_box
1171 \normalbaselines

```

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

```

1172 \bool_if:NT \l_@@_small_bool
1173 {
1174     \cs_set_nopar:Npn \arraystretch { 0.47 }
1175     \dim_set:Nn \arraycolsep { 1.45 pt }
1176 }

1177 \bool_if:NT \g_@@_recreate_cell_nodes_bool
1178 {
1179     \tl_put_right:Nn \@@_begin_of_row:
1180     {
1181         \pgfsys@markposition
1182         { \@@_env: - row - \int_use:N \c@iRow - base }
1183     }
1184 }

```

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`.

```

1185 \cs_set_nopar:Npn \ialign
1186 {
1187     \bool_if:NTF \c_@@_colortbl_loaded_bool

```

⁶⁵cf. `\nicematrix@redefine@check@rerun`

```

1188     {
1189         \CT@everycr
1190         {
1191             \noalign { \cs_gset_eq:NN \CT@row@color \prg_do_nothing: }
1192             \@@_everycr:
1193         }
1194     }
1195     { \everycr { \@@_everycr: } }
1196     \tabskip = \c_zero_skip

```

The box `\@arstrutbox` is a box constructed in the beginning of the environment `{array}`. The construction of that box takes into account the current value of `\arraystretch`⁶⁶ 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`.

```

1197     \dim_gzero_new:N \g_@@_dp_row_zero_dim
1198     \dim_gset:Nn \g_@@_dp_row_zero_dim { \box_dp:N \@arstrutbox }
1199     \dim_gzero_new:N \g_@@_ht_row_zero_dim
1200     \dim_gset:Nn \g_@@_ht_row_zero_dim { \box_ht:N \@arstrutbox }
1201     \dim_gzero_new:N \g_@@_ht_row_one_dim
1202     \dim_gset:Nn \g_@@_ht_row_one_dim { \box_ht:N \@arstrutbox }
1203     \dim_gzero_new:N \g_@@_dp_ante_last_row_dim
1204     \dim_gzero_new:N \g_@@_ht_last_row_dim
1205     \dim_gset:Nn \g_@@_ht_last_row_dim { \box_ht:N \@arstrutbox }
1206     \dim_gzero_new:N \g_@@_dp_last_row_dim
1207     \dim_gset:Nn \g_@@_dp_last_row_dim { \box_dp:N \@arstrutbox }

```

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

```

1208     \cs_set_eq:NN \ialign \@@_old_ialign:
1209     \halign
1210 }

```

We keep in memory the old versions of `\ldots`, `\cdots`, etc. only because we use them inside `\phantom` commands in order that the new commands `\Ldots`, `\Cdots`, etc. give the same spacing (except when the option `nullify-dots` is used).

```

1211 \cs_set_eq:NN \@@_old_ldots \ldots
1212 \cs_set_eq:NN \@@_old_cdots \cdots
1213 \cs_set_eq:NN \@@_old_vdots \vdots
1214 \cs_set_eq:NN \@@_old_ddots \ddots
1215 \cs_set_eq:NN \@@_old_iddots \iddots
1216 \bool_if:NTF \l_@@_standard_cline_bool
1217 { \cs_set_eq:NN \cline \@@_standard_cline }
1218 { \cs_set_eq:NN \cline \@@_cline }
1219 \cs_set_eq:NN \Ldots \@@_Ldots
1220 \cs_set_eq:NN \Cdots \@@_Cdots
1221 \cs_set_eq:NN \Vdots \@@_Vdots
1222 \cs_set_eq:NN \Ddots \@@_Ddots
1223 \cs_set_eq:NN \Iddots \@@_Iddots
1224 \cs_set_eq:NN \hdottedline \@@_hdottedline:
1225 \cs_set_eq:NN \Hline \@@_Hline:
1226 \cs_set_eq:NN \Hspace \@@_Hspace:
1227 \cs_set_eq:NN \Hdotsfor \@@_Hdotsfor:
1228 \cs_set_eq:NN \Vdotsfor \@@_Vdotsfor:
1229 \cs_set_eq:NN \Block \@@_Block:
1230 \cs_set_eq:NN \rotate \@@_rotate:
1231 \cs_set_eq:NN \OnlyMainNiceMatrix \@@_OnlyMainNiceMatrix:n
1232 \cs_set_eq:NN \dotfill \@@_old_dotfill:
1233 \cs_set_eq:NN \CodeAfter \@@_CodeAfter:

```

⁶⁶The option `small` of `nicematrix` changes (among others) the value of `\arraystretch`. This is done, of course, before the call of `{array}`.

```

1234 \cs_set_eq:NN \diagbox \@@_diagbox:nn
1235 \cs_set_eq:NN \NotEmpty \@@_NotEmpty:
1236 \cs_set_eq:NN \RowStyle \@@_RowStyle:n
1237 \bool_if:NT \l_@@_colortbl_like_bool \@@_colortbl_like:
1238 \bool_if:NT \l_@@_renew_dots_bool \@@_renew_dots:

```

We redefine `\multicolumn` and, since we want `\multicolumn` to be available in the potential environments `{tabular}` nested in the environments of `nicematrix`, we patch `{tabular}` to go back to the original definition.

```

1239 \cs_set_eq:NN \multicolumn \@@_multicolumn:nnn
1240 \hook_gput_code:nnn { env / tabular / begin } { . }
1241 { \cs_set_eq:NN \multicolumn \@@_old_multicolumn }

```

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

```

1242 \seq_gclear:N \g_@@_multicolumn_cells_seq
1243 \seq_gclear:N \g_@@_multicolumn_sizes_seq

```

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

```

1244 \int_gset:Nn \c@iRow { \l_@@_first_row_int - 1 }

```

At the end of the environment `{array}`, `\c@iRow` will be the total number de rows.

`\g_@@_row_total_int` will be the number or rows excepted the last row (if `\l_@@_last_row_bool` has been raised with the option `last-row`).

```

1245 \int_gzero_new:N \g_@@_row_total_int

```

The counter `\c@jCol` will be used to count the columns of the array. Since we want to know the total number of columns of the matrix, we also create a counter `\g_@@_col_total_int`. These counters are updated in the command `\@@_cell_begin:w` executed at the beginning of each cell.

```

1246 \int_gzero_new:N \g_@@_col_total_int
1247 \cs_set_eq:NN \@ifnextchar \new@ifnextchar
1248 \@@_renew_NC@rewrite@S:
1249 \bool_gset_false:N \g_@@_last_col_found_bool

```

During the construction of the array, the instructions `\Cdots`, `\Ldots`, etc. will be written in token lists `\g_@@_Cdots_lines_tl`, etc. which will be executed after the construction of the array.

```

1250 \tl_gclear_new:N \g_@@_Cdots_lines_tl
1251 \tl_gclear_new:N \g_@@_Ldots_lines_tl
1252 \tl_gclear_new:N \g_@@_Vdots_lines_tl
1253 \tl_gclear_new:N \g_@@_Ddots_lines_tl
1254 \tl_gclear_new:N \g_@@_Iddots_lines_tl
1255 \tl_gclear_new:N \g_@@_HVDotsfor_lines_tl

1256 \tl_gclear_new:N \g_nicematrix_code_before_tl
1257 }

```

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

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

```

1258 \cs_new_protected:Npn \@@_pre_array:
1259 {
1260 \cs_if_exist:NT \theiRow { \int_set_eq:NN \l_@@_old_iRow_int \c@iRow }
1261 \int_gzero_new:N \c@iRow
1262 \cs_if_exist:NT \thejCol { \int_set_eq:NN \l_@@_old_jCol_int \c@jCol }
1263 \int_gzero_new:N \c@jCol

```

We recall that `\l_@@_last_row_int` and `\l_@@_last_column_int` are *not* the numbers of the last row and last column of the array. There are only the values of the keys `last-row` and `last-column` (maybe the user has provided erroneous values). The meaning of that counters does not change during the environment of `nicematrix`. There is only a slight adjustment: if the user have used one of those keys without value, we provide now the right value as read on the `aux` file (of course, it's possible only after the first compilation).

```

1264 \int_compare:nNnT \l_@@_last_row_int = { -1 }
1265 {
1266   \bool_set_true:N \l_@@_last_row_without_value_bool
1267   \bool_if:NT \g_@@_aux_found_bool
1268     { \int_set:Nn \l_@@_last_row_int { \seq_item:Nn \c_@@_size_seq 3 } }
1269 }
1270 \int_compare:nNnT \l_@@_last_col_int = { -1 }
1271 {
1272   \bool_if:NT \g_@@_aux_found_bool
1273     { \int_set:Nn \l_@@_last_col_int { \seq_item:Nn \c_@@_size_seq 6 } }
1274 }

```

If there is a 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”.

```

1275 \int_compare:nNnT \l_@@_last_row_int > { -2 }
1276 {
1277   \tl_put_right:Nn \@@_update_for_first_and_last_row:
1278     {
1279       \dim_gset:Nn \g_@@_ht_last_row_dim
1280         { \dim_max:nn \g_@@_ht_last_row_dim { \box_ht:N \l_@@_cell_box } }
1281       \dim_gset:Nn \g_@@_dp_last_row_dim
1282         { \dim_max:nn \g_@@_dp_last_row_dim { \box_dp:N \l_@@_cell_box } }
1283     }
1284 }

1285 \seq_gclear:N \g_@@_cols_vlism_seq
1286 \seq_gclear:N \g_@@_submatrix_seq

```

Now the `\CodeBefore`.

```

1287 \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.

```

1288 \seq_gclear:N \g_@@_pos_of_blocks_seq

```

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

```

1289 \seq_gclear_new:N \g_@@_multicolumn_cells_seq
1290 \seq_gclear_new:N \g_@@_multicolumn_sizes_seq

```

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

```

1291 \@@_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.

```

1292 \box_clear_new:N \l_@@_the_array_box

```

The preamble will be constructed in `\g_@@_preamble_tl`.

```

1293 \@@_construct_preamble:

```

Now, the preamble is constructed in `\g_@@_preamble_tl`

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

```

1294 \dim_zero_new:N \l_@@_left_delim_dim
1295 \dim_zero_new:N \l_@@_right_delim_dim
1296 \bool_if:NTF \l_@@_NiceArray_bool
1297 {
1298   \dim_gset:Nn \l_@@_left_delim_dim { 2 \arraycolsep }
1299   \dim_gset:Nn \l_@@_right_delim_dim { 2 \arraycolsep }
1300 }
1301 {

```

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

```

1302 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_left_delim_tl $ }
1303 \dim_set:Nn \l_@@_left_delim_dim { \box_wd:N \l_tmpa_box }
1304 \hbox_set:Nn \l_tmpa_box { $ \bBigg@ 5 \g_@@_right_delim_tl $ }
1305 \dim_set:Nn \l_@@_right_delim_dim { \box_wd:N \l_tmpa_box }
1306 }

```

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).

```

1307 \hbox_set:Nw \l_@@_the_array_box
1308 \skip_horizontal:N \l_@@_left_margin_dim
1309 \skip_horizontal:N \l_@@_extra_left_margin_dim
1310 \c_math_toggle_token
1311 \bool_if:NTF \l_@@_light_syntax_bool
1312 { \use:c { @@-light-syntax } }
1313 { \use:c { @@-normal-syntax } }
1314 }

```

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

```

1315 \cs_new_protected:Npn \@@_pre_array_i:w #1 \Body
1316 {
1317   \tl_put_right:Nn \l_@@_code_before_tl { #1 }
1318   \bool_set_true:N \l_@@_code_before_bool

```

We go on with `\@@_pre_array:` which will (among other) execute the `\CodeBefore` (specified in the key `code-before` or after the keyword `\CodeBefore`). By definition, the `\CodeBefore` must be executed before the body of the array...

```

1319 \@@_pre_array:
1320 }

```

The `\CodeBefore`

The following command will be executed if the `\CodeBefore` has to be actually executed.

```

1321 \cs_new_protected:Npn \@@_pre_code_before:
1322 {

```

First, we give values to the LaTeX counters `iRow` and `jCol`. We remind that, in the `\CodeBefore` (and in the `\CodeAfter`) they represent the numbers of rows and columns of the array (without the potential last row and last column). The value of `\g_@@_row_total_int` is the number of the last row (with potentially a last exterior row) and `\g_@@_col_total_int` is the number of the last column (with potentially a last exterior column).

```

1323 \int_set:Nn \c@iRow { \seq_item:Nn \c_@@_size_seq 2 }
1324 \int_set:Nn \c@jCol { \seq_item:Nn \c_@@_size_seq 5 }
1325 \int_set_eq:NN \g_@@_row_total_int { \seq_item:Nn \c_@@_size_seq 3 }
1326 \int_set_eq:NN \g_@@_col_total_int { \seq_item:Nn \c_@@_size_seq 6 }

```

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

```

1327 \pgfsys@markposition { \@@_env: - position }
1328 \pgfsys@getposition { \@@_env: - position } \@@_picture_position:
1329 \pgfpicture
1330 \pgf@relevantforpicturesizefalse

```

First, the recreation of the `row` nodes.

```

1331 \int_step_inline:nnn \l_@@_first_row_int { \g_@@_row_total_int + 1 }
1332 {
1333   \pgfsys@getposition { \@@_env: - row - ##1 } \@@_node_position:
1334   \pgfcoordinate { \@@_env: - row - ##1 }
1335   { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1336 }

```

Now, the recreation of the `col` nodes.

```

1337 \int_step_inline:nnn \l_@@_first_col_int { \g_@@_col_total_int + 1 }
1338 {
1339   \pgfsys@getposition { \@@_env: - col - ##1 } \@@_node_position:
1340   \pgfcoordinate { \@@_env: - col - ##1 }
1341   { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1342 }

```

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

```

1343 \@@_create_diag_nodes:

```

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

```

1344 \bool_if:NT \g_@@_recreate_cell_nodes_bool \@@_recreate_cell_nodes:
1345 \endpgfpicture

```

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

```

1346 \@@_create_blocks_nodes:
1347 \bool_if:NT \c_@@_tikz_loaded_bool
1348 {
1349   \tikzset
1350   {
1351     every~picture / .style =
1352     { overlay , name~prefix = \@@_env: - }
1353   }
1354 }
1355 \cs_set_eq:NN \cellcolor \@@_cellcolor
1356 \cs_set_eq:NN \rectanglecolor \@@_rectanglecolor
1357 \cs_set_eq:NN \roundedrectanglecolor \@@_roundedrectanglecolor
1358 \cs_set_eq:NN \rowcolor \@@_rowcolor
1359 \cs_set_eq:NN \rowcolors \@@_rowcolors
1360 \cs_set_eq:NN \rowlistcolors \@@_rowlistcolors
1361 \cs_set_eq:NN \arraycolor \@@_arraycolor
1362 \cs_set_eq:NN \columncolor \@@_columncolor
1363 \cs_set_eq:NN \chessboardcolors \@@_chessboardcolors
1364 \cs_set_eq:NN \SubMatrix \@@_SubMatrix_in_code_before
1365 }

```

```

1366 \cs_new_protected:Npn \@@_exec_code_before:
1367 {
1368   \seq_gclear_new:N \g_@@_colors_seq
1369   \bool_gset_false:N \g_@@_recreate_cell_nodes_bool
1370   \group_begin:

```

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

```

1371 \bool_if:NT \l_@@_NiceTabular_bool \c_math_toggle_token

```

Here is the `\CodeBefore`. The construction is a bit complicated because `\l_@@_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 `\l_@@_code_before_tl` (when it is asked for the creation of cell nodes in the `\CodeBefore`). That's why we begin with a `\q_stop`: it will be used to discard the rest of `\l_@@_code_before_tl`.

```
1372 \exp_last_unbraced:NV \@@_CodeBefore_keys: \l_@@_code_before_tl \q_stop
```

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.

```
1373 \@@_actually_color:
1374 \bool_if:NT \l_@@_NiceTabular_bool \c_math_toggle_token
1375 \group_end:
1376 \bool_if:NT \g_@@_recreate_cell_nodes_bool
1377   { \tl_put_left:Nn \@@_node_for_cell: \@@_patch_node_for_cell: }
1378 }
```

```
1379 \keys_define:nn { NiceMatrix / CodeBefore }
1380 {
1381   create-cell-nodes .bool_gset:N = \g_@@_recreate_cell_nodes_bool ,
1382   create-cell-nodes .default:n = true ,
1383   sub-matrix .code:n = \keys_set:nn { NiceMatrix / sub-matrix } { #1 } ,
1384   sub-matrix .value_required:n = true ,
1385   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
1386   delimiters / color .value_required:n = true ,
1387   unknown .code:n = \@@_error:n { Unknown-key-for-CodeAfter }
1388 }

1389 \NewDocumentCommand \@@_CodeBefore_keys: { 0 { } }
1390 {
1391   \keys_set:nn { NiceMatrix / CodeBefore } { #1 }
1392   \@@_CodeBefore:w
1393 }
```

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 `\CodeAfter`, excepted, of course, if we are in the first compilation.

```
1394 \cs_new_protected:Npn \@@_CodeBefore:w #1 \q_stop
1395 {
1396   \bool_if:NT \g_@@_aux_found_bool
1397   {
1398     \@@_pre_code_before:
1399     #1
1400   }
1401 }
```

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.

```
1402 \cs_new_protected:Npn \@@_recreate_cell_nodes:
1403 {
1404   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
1405   {
1406     \pgfsys@getposition { \@@_env: - ##1 - base } \@@_node_position:
1407     \pgfcoordinate { \@@_env: - row - ##1 - base }
1408     { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1409     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
1410     {
1411       \cs_if_exist:cT
1412       { pgf @ sys @ pdf @ mark @ pos @ \@@_env: - ##1 - ####1 - NW }
```



```

1413         {
1414             \pgfsys@getposition
1415             { \@@_env: - ##1 - #####1 - NW }
1416             \@@_node_position:
1417             \pgfsys@getposition
1418             { \@@_env: - ##1 - #####1 - SE }
1419             \@@_node_position_i:
1420             \@@_pgf_rect_node:nnn
1421             { \@@_env: - ##1 - #####1 }
1422             { \pgfpointdiff \@@_picture_position: \@@_node_position: }
1423             { \pgfpointdiff \@@_picture_position: \@@_node_position_i: }
1424         }
1425     }
1426 }
1427 \int_step_inline:nn \c@iRow
1428 {
1429     \pgfnodealias
1430     { \@@_env: - ##1 - last }
1431     { \@@_env: - ##1 - \int_use:N \c@jCol }
1432 }
1433 \int_step_inline:nn \c@jCol
1434 {
1435     \pgfnodealias
1436     { \@@_env: - last - ##1 }
1437     { \@@_env: - \int_use:N \c@iRow - ##1 }
1438 }
1439 \@@_create_extra_nodes:
1440 }

```

```

1441 \cs_new_protected:Npn \@@_create_blocks_nodes:
1442 {
1443     \pgfpicture
1444     \pgf@relevantforpicturesizefalse
1445     \pgfrememberpicturepositiononpagetrue
1446     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
1447     { \@@_create_one_block_node:nnnnn ##1 }
1448     \endpgfpicture
1449 }

```

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.⁶⁷

```

1450 \cs_new_protected:Npn \@@_create_one_block_node:nnnnn #1 #2 #3 #4 #5
1451 {
1452     \tl_if_empty:nF { #5 }
1453     {
1454         \@@_qpoint:n { col - #2 }
1455         \dim_set_eq:NN \l_tmpa_dim \pgf@x
1456         \@@_qpoint:n { #1 }
1457         \dim_set_eq:NN \l_tmpb_dim \pgf@y
1458         \@@_qpoint:n { col - \@@_succ:n { #4 } }
1459         \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
1460         \@@_qpoint:n { \@@_succ:n { #3 } }
1461         \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
1462         \@@_pgf_rect_node:nnnnn
1463         { \@@_env: - #5 }
1464         { \dim_use:N \l_tmpa_dim }
1465         { \dim_use:N \l_tmpb_dim }
1466         { \dim_use:N \l_@@_tmpc_dim }
1467         { \dim_use:N \l_@@_tmpd_dim }

```

⁶⁷Moreover, 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).

```

1468   }
1469 }

1470 \cs_new_protected:Npn \@@_patch_for_revtext:
1471 {
1472   \cs_set_eq:NN \@addamp \@addamp@LaTeX
1473   \cs_set_eq:NN \insert@column \insert@column@array
1474   \cs_set_eq:NN \@classx \@classx@array
1475   \cs_set_eq:NN \@xarraycr \@xarraycr@array
1476   \cs_set_eq:NN \@arraycr \@arraycr@array
1477   \cs_set_eq:NN \@xargarraycr \@xargarraycr@array
1478   \cs_set_eq:NN \array \array@array
1479   \cs_set_eq:NN \@array \@array@array
1480   \cs_set_eq:NN \@tabular \@tabular@array
1481   \cs_set_eq:NN \mkpream \mkpream@array
1482   \cs_set_eq:NN \endarray \endarray@array
1483   \cs_set:Npn \@tabarray { \ifnextchar [ { \array } { \array [ c ] } }
1484   \cs_set:Npn \endtabular { \endarray $\egroup} % $
1485 }

```

The environment {NiceArrayWithDelims}

```

1486 \NewDocumentEnvironment { NiceArrayWithDelims }
1487 { m m O { } m ! O { } t \CodeBefore }
1488 {
1489   \bool_if:NT \c_@@_revtex_bool \@@_patch_for_revtext:
1490   \@@_provide_pgfsyspdfmark:
1491   \bool_if:NT \c_@@_footnote_bool \savenotes

```

The aim of the following `\bgroup` (the corresponding `\egroup` is, of course, at the end of the environment) is to be able to put an exponent to a matrix in a mathematical formula.

```

1492   \bgroup

1493   \tl_gset:Nn \g_@@_left_delim_tl { #1 }
1494   \tl_gset:Nn \g_@@_right_delim_tl { #2 }
1495   \tl_gset:Nn \g_@@_preamble_tl { #4 }

1496   \int_gzero:N \g_@@_block_box_int
1497   \dim_zero:N \g_@@_width_last_col_dim
1498   \dim_zero:N \g_@@_width_first_col_dim
1499   \bool_gset_false:N \g_@@_row_of_col_done_bool
1500   \str_if_empty:NT \g_@@_name_env_str
1501     { \str_gset:Nn \g_@@_name_env_str { NiceArrayWithDelims } }
1502   \bool_if:NTF \l_@@_NiceTabular_bool
1503     \mode_leave_vertical:
1504     \@@_test_if_math_mode:
1505   \bool_if:NT \l_@@_in_env_bool { \@@_fatal:n { Yet~in~env } }
1506   \bool_set_true:N \l_@@_in_env_bool

```

The command `\CT@arc@` contains the instruction of color for the rules of the array⁶⁸. 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.

```

1507   \cs_gset_eq:NN \@@_old_CT@arc@ \CT@arc@

```

⁶⁸e.g. `\color[rgb]{0.5,0.5,0}`

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.

```

1508 \cs_if_exist:NT \tikz@library@external@loaded
1509 {
1510     \tikzexternaldisable
1511     \cs_if_exist:NT \ifstandalone
1512     { \tikzset { external / optimize = false } }
1513 }

```

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

```

1514 \int_gincr:N \g_@@_env_int
1515 \bool_if:NF \l_@@_block_auto_columns_width_bool
1516 { \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`).

```

1517 \seq_gclear:N \g_@@_blocks_seq
1518 \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`.

```

1519 \seq_gclear:N \g_@@_pos_of_stroken_blocks_seq
1520 \seq_gclear:N \g_@@_pos_of_xdots_seq
1521 \tl_gclear_new:N \g_@@_code_before_tl
1522 \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.

```

1523 \bool_gset_false:N \g_@@_aux_found_bool
1524 \tl_if_exist:cT { c_@@ _ \int_use:N \g_@@_env_int _ tl }
1525 {
1526     \bool_gset_true:N \g_@@_aux_found_bool
1527     \use:c { c_@@ _ \int_use:N \g_@@_env_int _ tl }
1528 }

```

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.

```

1529 \tl_gclear:N \g_@@_aux_tl
1530 \tl_if_empty:NF \g_@@_code_before_tl
1531 {
1532     \bool_set_true:N \l_@@_code_before_bool
1533     \tl_put_right:NV \l_@@_code_before_tl \g_@@_code_before_tl
1534 }

```

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`.

```

1535 \bool_if:NTF \l_@@_NiceArray_bool
1536 { \keys_set:nn { NiceMatrix / NiceArray } }
1537 { \keys_set:nn { NiceMatrix / pNiceArray } }
1538 { #3 , #5 }

1539 \tl_if_empty:NF \l_@@_rules_color_tl
1540 { \exp_after:wN \@@_set_CT@arc@: \l_@@_rules_color_tl \q_stop }

```

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

```

1541 \IfBooleanTF { #6 } \@@_pre_array_i:w \@@_pre_array:

```

```

1542 }
1543 {
1544   \bool_if:NTF \l_@@_light_syntax_bool
1545     { \use:c { end @@-light-syntax } }
1546     { \use:c { end @@-normal-syntax } }
1547   \c_math_toggle_token
1548   \skip_horizontal:N \l_@@_right_margin_dim
1549   \skip_horizontal:N \l_@@_extra_right_margin_dim
1550   \hbox_set_end:

```

End of the construction of the array (in the box `\l_@@_the_array_box`).

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

```

1551   \bool_if:NT \l_@@_width_used_bool
1552   {
1553     \int_compare:nNnT \g_@@_total_X_weight_int = 0
1554       { \@@_error:n { width~without~X~columns } }
1555   }

```

Now, if there is at least one `X`-column in the environment, we compute the width that those columns will have (in the next compilation). In fact, `\l_@@_X_columns_dim` will be the width of a column of weight 1. For a `X`-column of weight n , the width will be `\l_@@_X_columns_dim` multiplied by n .

```

1556   \int_compare:nNnT \g_@@_total_X_weight_int > 0
1557   {
1558     \tl_gput_right:Nx \g_@@_aux_tl
1559     {
1560       \bool_set_true:N \l_@@_X_columns_aux_bool
1561       \dim_set:Nn \l_@@_X_columns_dim
1562       {
1563         \dim_compare:nNnTF
1564           {
1565             \dim_abs:n
1566             { \l_@@_width_dim - \box_wd:N \l_@@_the_array_box }
1567           }
1568           <
1569           { 0.001 pt }
1570           { \dim_use:N \l_@@_X_columns_dim }
1571           {
1572             \dim_eval:n
1573             {
1574               ( \l_@@_width_dim - \box_wd:N \l_@@_the_array_box )
1575               / \int_use:N \g_@@_total_X_weight_int
1576               + \l_@@_X_columns_dim
1577             }
1578           }
1579       }
1580     }
1581   }

```

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

```

1582   \int_compare:nNnT \l_@@_last_row_int > { -2 }
1583   {
1584     \bool_if:NF \l_@@_last_row_without_value_bool
1585     {
1586       \int_compare:nNnF \l_@@_last_row_int = \c@iRow
1587       {
1588         \@@_error:n { Wrong~last~row }
1589         \int_gset_eq:NN \l_@@_last_row_int \c@iRow
1590       }
1591     }
1592   }

```

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”.⁶⁹

```

1593 \int_gset_eq:NN \c@jCol \g_@@_col_total_int
1594 \bool_if:nTF \g_@@_last_col_found_bool
1595   { \int_gdecr:N \c@jCol }
1596   {
1597     \int_compare:nNnT \l_@@_last_col_int > { -1 }
1598     { \@@_error:n { last~col~not~used } }
1599   }

```

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

```

1600 \int_gset_eq:NN \g_@@_row_total_int \c@iRow
1601 \int_compare:nNnT \l_@@_last_row_int > { -1 } { \int_gdecr:N \c@iRow }

```

Now, we begin the real construction in the output flow of TeX. First, we take into account a potential “first column” (we remind that this “first column” has been constructed in an overlapping position and that we have computed its width in `\g_@@_width_first_col_dim`: see p. 128).

```

1602 \int_compare:nNnT \l_@@_first_col_int = 0
1603   {
1604     \skip_horizontal:N \col@sep
1605     \skip_horizontal:N \g_@@_width_first_col_dim
1606   }

```

The construction of the real box is different when `\l_@@_NiceArray_bool` is true (`{NiceArray}` or `{NiceTabular}`) and in the other environments because, in `{NiceArray}` or `{NiceTabular}`, we have no delimiter to put (but we have tabular notes to put). We begin with this case.

```

1607 \bool_if:nTF \l_@@_NiceArray_bool
1608   {
1609     \str_case:VnF \l_@@_baseline_tl
1610     {
1611       b \@@_use_arraybox_with_notes_b:
1612       c \@@_use_arraybox_with_notes_c:
1613     }
1614     \@@_use_arraybox_with_notes:
1615   }

```

Now, in the case of an environment `{pNiceArray}`, `{bNiceArray}`, etc. We compute `\l_tmpa_dim` which is the total height of the “first row” above the array (when the key `first-row` is used).

```

1616   {
1617     \int_compare:nNnTF \l_@@_first_row_int = 0
1618     {
1619       \dim_set_eq:NN \l_tmpa_dim \g_@@_dp_row_zero_dim
1620       \dim_add:Nn \l_tmpa_dim \g_@@_ht_row_zero_dim
1621     }
1622     { \dim_zero:N \l_tmpa_dim }

```

We compute `\l_tmpb_dim` which is the total height of the “last row” below the array (when the key `last-row` is used). A value of `-2` for `\l_@@_last_row_int` means that there is no “last row”.⁷⁰

```

1623     \int_compare:nNnTF \l_@@_last_row_int > { -2 }
1624     {
1625       \dim_set_eq:NN \l_tmpb_dim \g_@@_ht_last_row_dim
1626       \dim_add:Nn \l_tmpb_dim \g_@@_dp_last_row_dim
1627     }
1628     { \dim_zero:N \l_tmpb_dim }
1629     \hbox_set:Nn \l_tmpa_box
1630     {
1631       \c_math_toggle_token
1632       \tl_if_empty:NF \l_@@_delimiters_color_tl
1633       { \color { \l_@@_delimiters_color_tl } }

```

⁶⁹We remind that the potential “first column” (exterior) has the number 0.

⁷⁰A value of `-1` for `\l_@@_last_row_int` means that there is a “last row” but the the user have not set the value with the option `last row` (and we are in the first compilation).

```

1634         \exp_after:wN \left \g_@@_left_delim_tl
1635         \vcenter
1636         {

```

We take into account the “first row” (we have previously computed its total height in `\l_tmpa_dim`). The `\hbox:n` (or `\hbox`) is necessary here. There was a bug in the following line (corrected the 2021/11/23).

```

1637         \skip_vertical:n { -\l_tmpa_dim - \arrayrulewidth }
1638         \hbox
1639         {
1640             \bool_if:NTF \l_@@_NiceTabular_bool
1641             { \skip_horizontal:N -\tabcolsep }
1642             { \skip_horizontal:N -\arraycolsep }
1643             \@@_use_arraybox_with_notes_c:
1644             \bool_if:NTF \l_@@_NiceTabular_bool
1645             { \skip_horizontal:N -\tabcolsep }
1646             { \skip_horizontal:N -\arraycolsep }
1647         }

```

We take into account the “last row” (we have previously computed its total height in `\l_tmpb_dim`). There was a bug in the following line (corrected the 2021/11/23).

```

1648         \skip_vertical:n { -\l_tmpb_dim + \arrayrulewidth }
1649     }

```

Curiously, we have to put again the following specification of color. Otherwise, with XeLaTeX (and not with the other engines), the closing delimiter is not colored.

```

1650         \tl_if_empty:NF \l_@@_delimiters_color_tl
1651         { \color { \l_@@_delimiters_color_tl } }
1652         \exp_after:wN \right \g_@@_right_delim_tl
1653         \c_math_toggle_token
1654     }

```

Now, the box `\l_tmpa_box` is created with the correct delimiters.

We will put the box in the TeX flow. However, we have a small work to do when the option `delimiters/max-width` is used.

```

1655         \bool_if:NTF \l_@@_delimiters_max_width_bool
1656         {
1657             \@@_put_box_in_flow_bis:nn
1658             \g_@@_left_delim_tl \g_@@_right_delim_tl
1659         }
1660         \@@_put_box_in_flow:
1661     }

```

We take into account a potential “last column” (this “last column” has been constructed in an overlapping position and we have computed its width in `\g_@@_width_last_col_dim`: see p. 129).

```

1662         \bool_if:NT \g_@@_last_col_found_bool
1663         {
1664             \skip_horizontal:N \g_@@_width_last_col_dim
1665             \skip_horizontal:N \col@sep
1666         }
1667         \bool_if:NF \l_@@_Matrix_bool
1668         {
1669             \int_compare:nNnT \c@jCol < \g_@@_static_num_of_col_int
1670             { \@@_error:n { columns-not-used } }
1671         }
1672         \group_begin:
1673         \globaldefs = 1
1674         \@@_msg_redirect_name:nn { columns-not-used } { error }
1675         \group_end:
1676         \@@_after_array:

```

The aim of the following `\egroup` (the corresponding `\bgroup` is, of course, at the beginning of the environment) is to be able to put an exposant to a matrix in a mathematical formula.

```

1677         \egroup

```

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

```

1678 \iow_now:Nn \@mainaux { \ExplSyntaxOn }
1679 \iow_now:Nn \@mainaux { \char_set_catcode_space:n { 32 } }
1680 \iow_now:Nx \@mainaux
1681 {
1682   \tl_gset:cn { c_@@_ \int_use:N \g_@@_env_int _tl }
1683   { \exp_not:V \g_@@_aux_tl }
1684 }
1685 \iow_now:Nn \@mainaux { \ExplSyntaxOff }

1686 \bool_if:NT \c_@@_footnote_bool \endsavenotes
1687 }

```

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

We construct the preamble of the array

The transformation of the preamble is an operation in several steps.

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

```

1688 \cs_new_protected:Npn \@@_construct_preamble:
1689 {

```

First, we will do an “expansion” of the preamble with the tools of the package `array` itself. This “expansion” will expand all the constructions with `*` and with all column types (defined by the user or by various packages using `\newcolumntype`).

Since we use the tools of `array` to do this expansion, we will have a programming which is not in the style of the L3 programming layer.

We redefine the column types `w` and `W`. We use `\@@_newcolumntype` instead of `\newcolumntype` because we don’t want warnings for column types already defined. These redefinitions are in fact *protections* of the letters `w` and `W`. We don’t want these columns type expanded because we will do the patch ourselves after. We want to be able to use the standard column types `w` and `W` in potential `{tabular}` of `array` in some cells of our array. That’s why we do those redefinitions in a TeX group.

```

1690 \group_begin:

```

If we are in an environment without explicit preamble, we have nothing to do (excepted the treatment on both sides of the preamble which will be done at the end).

```

1691 \bool_if:NF \l_@@_Matrix_bool
1692 {
1693   \@@_newcolumntype w [ 2 ] { \@@_w: { ##1 } { ##2 } }
1694   \@@_newcolumntype W [ 2 ] { \@@_W: { ##1 } { ##2 } }

```

If the package `varwidth` has defined the column type `V`, we protect from expansion by redefining it to `\@@_V:` (which will be caught by our system).

```

1695 \cs_if_exist:NT \NC@find@V { \@@_newcolumntype V { \@@_V: } }

```

First, we have to store our preamble in the token register `\@temptokena` (those “token registers” are *not* supported by the L3 programming layer).

```

1696 \exp_args:NV \@temptokena \g_@@_preamble_tl

```

Initialisation of a flag used by `array` to detect the end of the expansion.

```

1697 \@tempswatruue

```

The following line actually does the expansion (it’s has been copied from `array.sty`). The expanded version is still in `\@temptokena`.

```

1698 \@whilesw \if@tempswa \fi { \@tempswafalse \the \NC@list }

```

Now, we have to “patch” that preamble by transforming some columns. We will insert in the TeX flow the preamble in its actual form (that is to say after the “expansion”) following by a marker `\q_stop` and we will consume these tokens constructing the (new form of the) preamble in `\g_@@_preamble_tl`. This is done recursively with the command `\@@_patch_preamble:n`. In the same time, we will count the columns with the counter `\c@jCol`.

```

1699     \int_gzero:N \c@jCol
1700     \tl_gclear:N \g_@@_preamble_tl
\g_tmpb_bool will be raised if you have a | at the end of the preamble.
1701     \bool_gset_false:N \g_tmpb_bool
1702     \tl_if_eq:NnTF \l_@@_vlines_clist { all }
1703     {
1704         \tl_gset:Nn \g_@@_preamble_tl
1705         { ! { \skip_horizontal:N \arrayrulewidth } }
1706     }
1707     {
1708         \clist_if_in:NnT \l_@@_vlines_clist 1
1709         {
1710             \tl_gset:Nn \g_@@_preamble_tl
1711             { ! { \skip_horizontal:N \arrayrulewidth } }
1712         }
1713     }

```

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

```

1714     \seq_clear:N \g_@@_cols_vlsim_seq

```

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

```

1715     \int_zero:N \l_tmpa_int

```

Now, we actually patch the preamble (and it is constructed in `\g_@@_preamble_tl`).

```

1716     \exp_after:wN \@@_patch_preamble:n \the \temptokena \q_stop
1717     \int_gset_eq:NN \g_@@_static_num_of_col_int \c@jCol
1718 }

```

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

```

1719     \bool_if:NT \l_@@_colortbl_like_bool
1720     {
1721         \regex_replace_all:NnN
1722         \c_@@_columncolor_regex
1723         { \c { @@_columncolor_preamble } }
1724         \g_@@_preamble_tl
1725     }

```

Now, we can close the TeX group which was opened for the redefinition of the columns of type `w` and `W`.

```

1726     \group_end:

```

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

```

1727     \bool_lazy_or:nnT
1728     { ! \str_if_eq_p:Nn \g_@@_left_delim_tl { . } }
1729     { ! \str_if_eq_p:Nn \g_@@_right_delim_tl { . } }
1730     { \bool_set_false:N \l_@@_NiceArray_bool }

```

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

```

1731     \bool_if:NT \g_tmpb_bool { \bool_set_true:N \l_@@_bar_at_end_of_pream_bool }

```

We complete the preamble with the potential “exterior columns” (on both sides).

```

1732     \int_compare:nNnTF \l_@@_first_col_int = 0
1733     { \tl_gput_left:NV \g_@@_preamble_tl \c_@@_preamble_first_col_tl }
1734     {
1735         \bool_lazy_all:nT

```



```

1736     {
1737         \l_@@_NiceArray_bool
1738         { \bool_not_p:n \l_@@_NiceTabular_bool }
1739         { \tl_if_empty_p:N \l_@@_vlines_clist }
1740         { \bool_not_p:n \l_@@_exterior_arraycolsep_bool }
1741     }
1742     { \tl_gput_left:Nn \g_@@_preamble_tl { @ { } } }
1743 }
1744 \int_compare:nNnTF \l_@@_last_col_int > { -1 }
1745 { \tl_gput_right:NV \g_@@_preamble_tl \c_@@_preamble_last_col_tl }
1746 {
1747     \bool_lazy_all:nT
1748     {
1749         \l_@@_NiceArray_bool
1750         { \bool_not_p:n \l_@@_NiceTabular_bool }
1751         { \tl_if_empty_p:N \l_@@_vlines_clist }
1752         { \bool_not_p:n \l_@@_exterior_arraycolsep_bool }
1753     }
1754     { \tl_gput_right:Nn \g_@@_preamble_tl { @ { } } }
1755 }

```

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*}` (`\l_@@_tabular_width_dim=0pt`).

```

1756     \dim_compare:nNnT \l_@@_tabular_width_dim = \c_zero_dim
1757     {
1758         \tl_gput_right:Nn \g_@@_preamble_tl
1759         { > { \@@_error_too_much_cols: } 1 }
1760     }
1761 }

```

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

```

1762 \cs_new_protected:Npn \@@_patch_preamble:n #1
1763 {
1764     \str_case:nnF { #1 }
1765     {
1766         c { \@@_patch_preamble_i:n #1 }
1767         l { \@@_patch_preamble_i:n #1 }
1768         r { \@@_patch_preamble_i:n #1 }
1769         > { \@@_patch_preamble_ii:nn #1 }
1770         ! { \@@_patch_preamble_ii:nn #1 }
1771         @ { \@@_patch_preamble_ii:nn #1 }
1772         | { \@@_patch_preamble_iii:n #1 }
1773         p { \@@_patch_preamble_iv:n #1 }
1774         b { \@@_patch_preamble_iv:n #1 }
1775         m { \@@_patch_preamble_iv:n #1 }
1776         \@@_V: { \@@_patch_preamble_v:n }
1777         V { \@@_patch_preamble_v:n }
1778         \@@_w: { \@@_patch_preamble_vi:nnnn { } #1 }
1779         \@@_W: { \@@_patch_preamble_vi:nnnn { \cs_set_eq:NN \hss \hfil } #1 }
1780         \@@_S: { \@@_patch_preamble_vii:n }
1781         ( { \@@_patch_preamble_viii:nn #1 }
1782         [ { \@@_patch_preamble_viii:nn #1 }
1783         \{ { \@@_patch_preamble_viii:nn #1 }
1784         ) { \@@_patch_preamble_ix:nn #1 }
1785         ] { \@@_patch_preamble_ix:nn #1 }
1786         \} { \@@_patch_preamble_ix:nn #1 }
1787         X { \@@_patch_preamble_x:n }

```

When `tabularx` is loaded, a local redefinition of the specifier `X` is done to replace `X` by `\@@_X`. Thus, our column type `X` will be used in the `{NiceTabularX}`.

```

1788     \@@_X { \@@_patch_preamble_x:n }

```

```

1789     \q_stop { }
1790 }
1791 {
1792   \str_case_e:nnF { #1 }
1793   {
1794     \l_@@_letter_for_dotted_lines_str { \@@_patch_preamble_xii:n #1 }
1795     \l_@@_letter_vlism_tl
1796     {
1797       \seq_gput_right:Nx \g_@@_cols_vlism_seq
1798       { \int_eval:n { \c@jCol + 1 } }
1799       \tl_gput_right:Nx \g_@@_preamble_tl
1800       { \exp_not:N ! { \skip_horizontal:N \arrayrulewidth } }
1801       \@@_patch_preamble:n
1802     }
1803   } : }
1804   {
1805     \bool_if:NTF \c_@@_arydshln_loaded_bool
1806     {
1807       \tl_gput_right:Nn \g_@@_preamble_tl { : }
1808       \@@_patch_preamble:n
1809     }
1810     { \@@_fatal:n { colon-without-arydshln } }
1811   }
1812 }

```

Now the case of a letter set by the final user for a customized rule. Such customized rule is defined by using the key `custom-line` in `\NiceMatrixOptions`. That key takes in as value a list of *key=value* pairs. Among the keys available in that list, there is the key `letter`. All the letters defined by this way by the final user for such customized rules are added in the set of keys `{NiceMatrix/ColumnTypes}`. That set of keys is used to store the characteristics of those types of rules for convenience: the keys of that set of keys won't never be used as keys by the final user (he will use, instead, letters in the preamble of its array).

```

1813   {
1814     \keys_set_known:nnN { NiceMatrix / ColumnTypes } { #1 } \l_tmpa_tl
1815     \tl_if_empty:NTF \l_tmpa_tl
1816     \@@_patch_preamble:n
1817     { \@@_fatal:nn { unknown-column-type } { #1 } }
1818   }
1819 }
1820 }

```

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

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

```

1821 \cs_new_protected:Npn \@@_patch_preamble_i:n #1
1822 {
1823   \tl_gput_right:Nn \g_@@_preamble_tl
1824   {
1825     > { \@@_cell_begin:w \str_set:Nn \l_@@_hpos_cell_str { #1 } }
1826     #1
1827     < \@@_cell_end:
1828   }

```

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

```

1829   \int_gincr:N \c@jCol
1830   \@@_patch_preamble_xi:n
1831 }

```

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

```

1832 \cs_new_protected:Npn \@@_patch_preamble_ii:nn #1 #2
1833 {
1834   \tl_gput_right:Nn \g_@@_preamble_tl { #1 { #2 } }

```

```

1835 \@@_patch_preamble:n
1836 }

```

For |

```

1837 \cs_new_protected:Npn \@@_patch_preamble_iii:n #1
1838 {

```

\l_tmpa_int is the number of successive occurrences of |

```

1839 \int_incr:N \l_tmpa_int
1840 \@@_patch_preamble_iii_i:n
1841 }

```

```

1842 \cs_new_protected:Npn \@@_patch_preamble_iii_i:n #1
1843 {
1844 \str_if_eq:nnTF { #1 } |
1845 { \@@_patch_preamble_iii:n | }
1846 {
1847 \tl_gput_right:Nx \g_@@_preamble_tl
1848 {
1849 \exp_not:N !
1850 {
1851 \skip_horizontal:n
1852 {

```

Here, the command \dim_eval:n is mandatory.

```

1853 \dim_eval:n
1854 {
1855 \arrayrulewidth * \l_tmpa_int
1856 + \doublerulesep * ( \l_tmpa_int - 1)
1857 }
1858 }
1859 }
1860 }
1861 \tl_gput_right:Nx \g_@@_internal_code_after_tl
1862 {
1863 \@@_vline:n
1864 {
1865 position = \@@_succ:n \c@jCol ,
1866 multiplicity = \int_use:N \l_tmpa_int ,
1867 }

```

We don't have provided value for **start** nor for **end**, which means that the rule will cover (potentially) all the rows of the array.

```

1868 }
1869 \int_zero:N \l_tmpa_int
1870 \str_if_eq:nnT { #1 } { \q_stop } { \bool_gset_true:N \g_tmpb_bool }
1871 \@@_patch_preamble:n #1
1872 }
1873 }
1874 \bool_new:N \l_@@_bar_at_end_of_pream_bool

```

The specifier **p** (and also the specifiers **m** and **b**) have an optional argument between square brackets for a list of *key-value* pairs. Here are the corresponding keys. This set of keys will also be used by the **X** columns.

```

1875 \keys_define:nn { WithArrows / p-column }
1876 {
1877 r .code:n = \str_set:Nn \l_@@_hpos_col_str { r } ,
1878 r .value_forbidden:n = true ,
1879 c .code:n = \str_set:Nn \l_@@_hpos_col_str { c } ,
1880 c .value_forbidden:n = true ,
1881 l .code:n = \str_set:Nn \l_@@_hpos_col_str { l } ,
1882 l .value_forbidden:n = true ,
1883 si .code:n = \str_set:Nn \l_@@_hpos_col_str { si } ,

```

```

1884     si .value_forbidden:n = true ,
1885     p .code:n = \str_set:Nn \l_@@_vpos_col_str { p } ,
1886     p .value_forbidden:n = true ,
1887     t .meta:n = p ,
1888     m .code:n = \str_set:Nn \l_@@_vpos_col_str { m } ,
1889     m .value_forbidden:n = true ,
1890     b .code:n = \str_set:Nn \l_@@_vpos_col_str { b } ,
1891     b .value_forbidden:n = true ,
1892 }

```

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

```

1893 \cs_new_protected:Npn \@@_patch_preamble_iv:n #1
1894 {
1895     \str_set:Nn \l_@@_vpos_col_str { #1 }

```

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

```

1896     \@@_patch_preamble_iv_i:n
1897 }

1898 \cs_new_protected:Npn \@@_patch_preamble_iv_i:n #1
1899 {
1900     \str_if_eq:nnTF { #1 } { [ ]
1901         { \@@_patch_preamble_iv_ii:w [ ]
1902           { \@@_patch_preamble_iv_ii:w [ ] { #1 } }
1903         }
1904     \cs_new_protected:Npn \@@_patch_preamble_iv_ii:w [ #1 ]
1905     { \@@_patch_preamble_iv_iii:nn { #1 } }

```

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

`#2` is the mandatory argument of the specifier: the width of the column.

```

1906 \cs_new_protected:Npn \@@_patch_preamble_iv_iii:nn #1 #2
1907 {

```

The possible values of `\l_@@_hpos_col_str` are `j` (for *justified* which is the initial value), `l`, `c` and `r` (when the user has used the corresponding key in the optional argument of the specifier).

```

1908     \str_set:Nn \l_@@_hpos_col_str { j }
1909     \keys_set:nn { WithArrows / p-column } { #1 }
1910     \@@_patch_preamble_iv_iv:nn { #2 } { minipage }
1911 }

```

The first argument is the width of the column. The second is the type of environment: `minipage` or `varwidth`.

```

1912 \cs_new_protected:Npn \@@_patch_preamble_iv_iv:nn #1 #2
1913 {
1914     \use:x
1915     {
1916         \@@_patch_preamble_iv_v:nnnnnnnn
1917         { \str_if_eq:VnTF \l_@@_vpos_col_str { p } { t } { b } }
1918         { \dim_eval:n { #1 } }
1919     }

```

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

```

1920         \str_if_eq:VnTF \l_@@_hpos_col_str j
1921         { \str_set:Nn \exp_not:N \l_@@_hpos_cell_str { c } }
1922         {
1923             \str_set:Nn \exp_not:N \l_@@_hpos_cell_str
1924             { \l_@@_hpos_col_str }
1925         }
1926     \str_case:Vn \l_@@_hpos_col_str
1927     {
1928         c { \exp_not:N \centering }

```

```

1929         l { \exp_not:N \raggedright }
1930         r { \exp_not:N \raggedleft }
1931     }
1932 }
1933 { \str_if_eq:VnT \l_@@_vpos_col_str { m } \@@_center_cell_box: }
1934 { \str_if_eq:VnT \l_@@_hpos_col_str { si } \siunitx_cell_begin:w }
1935 { \str_if_eq:VnT \l_@@_hpos_col_str { si } \siunitx_cell_end: }
1936 { #2 }
1937 {
1938     \str_case:VnF \l_@@_hpos_col_str
1939     {
1940         { j } { c }
1941         { si } { c }
1942     }
1943     { \l_@@_hpos_col_str }
1944 }
1945 }

```

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

```

1946     \int_gincr:N \c@jCol
1947     \@@_patch_preamble_xi:n
1948 }

```

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

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

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

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

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

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

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

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

```

1949 \cs_new_protected:Npn \@@_patch_preamble_iv_v:nnnnnnnn #1 #2 #3 #4 #5 #6 #7 #8
1950 {
1951     \tl_gput_right:Nn \g_@@_preamble_tl
1952     {
1953         > {

```

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.

```

1954         \dim_set:Nn \l_@@_col_width_dim { #2 }
1955         \@@_cell_begin:w
1956         \begin { #7 } [ #1 ] { #2 }

```

The following lines have been taken from array.sty.

```

1957     \everypar
1958     {
1959         \vrule height \box_ht:N \@arstrutbox width \c_zero_dim
1960         \everypar { }
1961     }

```

Now, the potential code for the horizontal position of the content of the cell (\centering, \raggedright, \raggedleft or nothing).

```

1962     #3

```

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

```

1963     \g_@@_row_style_tl
1964     \arraybackslash
1965     #5
1966 }

```

```

1967         #8
1968     < {
1969         #6

```

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

```

1970         \@finalstrut \@arstrutbox
1971         % \bool_if:NT \g_@@_rotate_bool { \raggedright \hsize = 3 cm }
1972         \end { #7 }

```

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

```

1973         #4
1974         \@@_cell_end:
1975     }
1976 }
1977 }

```

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

```

1978 \cs_new_protected:Npn \@@_center_cell_box:
1979 {

```

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

```

1980     \tl_gput_right:Nn \g_@@_post_action_cell_tl
1981     {
1982         \int_compare:nNnT
1983         { \box_ht:N \l_@@_cell_box }
1984         >
1985         { \box_ht:N \@arstrutbox }
1986         {
1987             \hbox_set:Nn \l_@@_cell_box
1988             {
1989                 \box_move_down:nn
1990                 {
1991                     ( \box_ht:N \l_@@_cell_box - \box_ht:N \@arstrutbox
1992                     + \baselineskip ) / 2
1993                 }
1994                 { \box_use:N \l_@@_cell_box }
1995             }
1996         }
1997     }
1998 }

```

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

```

1999 \cs_new_protected:Npn \@@_patch_preamble_v:n #1
2000 {
2001     \str_if_eq:nnTF { #1 } { [ ] }
2002     { \@@_patch_preamble_v_i:w [ ] }
2003     { \@@_patch_preamble_v_i:w [ ] { #1 } }
2004 }
2005 \cs_new_protected:Npn \@@_patch_preamble_v_i:w [ #1 ]
2006 { \@@_patch_preamble_v_ii:nn { #1 } }
2007 \cs_new_protected:Npn \@@_patch_preamble_v_ii:nn #1 #2
2008 {
2009     \str_set:Nn \l_@@_vpos_col_str { p }
2010     \str_set:Nn \l_@@_hpos_col_str { j }
2011     \keys_set:nn { WithArrows / p-column } { #1 }
2012     \bool_if:NTF \c_@@_varwidth_loaded_bool
2013     { \@@_patch_preamble_iv_iv:nn { #2 } { varwidth } }
2014     {

```

```

2015     \@@_error:n { varwidth~not~loaded }
2016     \@@_patch_preamble_iv_iv:nn { #2 } { minipage }
2017   }
2018 }

```

For w and W

```

2019 \cs_new_protected:Npn \@@_patch_preamble_vi:nnnn #1 #2 #3 #4
2020 {
2021   \tl_gput_right:Nn \g_@@_preamble_tl
2022   {
2023     > {

```

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.

```

2024       \dim_set:Nn \l_@@_col_width_dim { #4 }
2025       \hbox_set:Nw \l_@@_cell_box
2026       \@@_cell_begin:w
2027       \str_set:Nn \l_@@_hpos_cell_str { #3 }
2028     }
2029   c
2030   < {
2031     \@@_cell_end:
2032     #1
2033     \hbox_set_end:
2034     \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
2035     \@@_adjust_size_box:
2036     \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2037   }
2038 }

```

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

```

2039   \int_gincr:N \c@jCol
2040   \@@_patch_preamble_xi:n
2041 }

```

For `\@@_S:`. If the user has used `S[...]`, `S` has been replaced by `\@@_S:` during the first expansion of the preamble (done with the tools of standard LaTeX and array).

```

2042 \cs_new_protected:Npn \@@_patch_preamble_vii:n #1
2043 {
2044   \str_if_eq:nnTF { #1 } { [ ] }
2045   { \@@_patch_preamble_vii_i:w [ ] }
2046   { \@@_patch_preamble_vii_i:w [ ] { #1 } }
2047 }
2048 \cs_new_protected:Npn \@@_patch_preamble_vii_i:w [ #1 ]
2049 { \@@_patch_preamble_vii_ii:n { #1 } }
2050 \cs_new_protected:Npn \@@_patch_preamble_vii_ii:n #1
2051 {

```

We test whether the version of nicematrix is at least 3.0. We will change the programming of the test further with something like `\VersionAtLeast`.

```

2052   \cs_if_exist:NTF \siunitx_cell_begin:w
2053   {
2054     \tl_gput_right:Nn \g_@@_preamble_tl
2055     {
2056       > {
2057         \@@_cell_begin:w
2058         \keys_set:nn { siunitx } { #1 }
2059         \siunitx_cell_begin:w
2060       }
2061       c
2062       < { \siunitx_cell_end: \@@_cell_end: }
2063     }

```

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

```

2064     \int_gincr:N \c@jCol
2065     \@@_patch_preamble_xi:n
2066   }
2067   { \@@_fatal:n { Version~of~siunitx~too~old } }
2068 }

```

For (, [and \{.

```

2069 \cs_new_protected:Npn \@@_patch_preamble_viii:nn #1 #2
2070 {
2071   \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.

```

2072   \int_compare:nNnTF \c@jCol = \c_zero_int
2073   {
2074     \str_if_eq:VnTF \g_@@_left_delim_tl { . }
2075     {

```

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

```

2076       \tl_gset:Nn \g_@@_left_delim_tl { #1 }
2077       \tl_gset:Nn \g_@@_right_delim_tl { . }
2078       \@@_patch_preamble:n #2
2079     }
2080     {
2081       \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } }
2082       \@@_patch_preamble_viii_i:nn { #1 } { #2 }
2083     }
2084   }
2085   { \@@_patch_preamble_viii_i:nn { #1 } { #2 } }
2086 }
2087 \cs_new_protected:Npn \@@_patch_preamble_viii_i:nn #1 #2
2088 {
2089   \tl_gput_right:Nx \g_@@_internal_code_after_tl
2090   { \@@_delimiter:nnn #1 { \@@_succ:n \c@jCol } \c_true_bool }
2091   \tl_if_in:nnTF { ( [ \{ ) ] \} } { #2 }
2092   {
2093     \@@_error:nn { delimiter~after~opening } { #2 }
2094     \@@_patch_preamble:n
2095   }
2096   { \@@_patch_preamble:n #2 }
2097 }

```

For),] and \}. 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}).

```

2098 \cs_new_protected:Npn \@@_patch_preamble_ix:nn #1 #2
2099 {
2100   \bool_if:NT \l_@@_small_bool { \@@_fatal:n { Delimiter~with~small } }
2101   \tl_if_in:nnTF { ) ] \} } { #2 }
2102   { \@@_patch_preamble_ix_i:nnn #1 #2 }
2103   {
2104     \tl_if_eq:nnTF { \q_stop } { #2 }
2105     {
2106       \str_if_eq:VnTF \g_@@_right_delim_tl { . }
2107       { \tl_gset:Nn \g_@@_right_delim_tl { #1 } }
2108       {
2109         \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } }
2110         \tl_gput_right:Nx \g_@@_internal_code_after_tl
2111         { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2112         \@@_patch_preamble:n #2

```



```

2113     }
2114   }
2115   {
2116     \tl_if_in:nnT { ( [ \{ } { #2 }
2117       { \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } } }
2118     \tl_gput_right:Nx \g_@@_internal_code_after_tl
2119       { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2120     \@@_patch_preamble:n #2
2121   }
2122 }
2123 }
2124 \cs_new_protected:Npn \@@_patch_preamble_ix_i:nnn #1 #2 #3
2125 {
2126   \tl_if_eq:nnTF { \q_stop } { #3 }
2127   {
2128     \str_if_eq:VnTF \g_@@_right_delim_tl { . }
2129     {
2130       \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } }
2131       \tl_gput_right:Nx \g_@@_internal_code_after_tl
2132         { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2133       \tl_gset:Nn \g_@@_right_delim_tl { #2 }
2134     }
2135     {
2136       \tl_gput_right:Nn \g_@@_preamble_tl { ! { \enskip } }
2137       \tl_gput_right:Nx \g_@@_internal_code_after_tl
2138         { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2139       \@@_error:nn { double-closing-delimiter } { #2 }
2140     }
2141   }
2142   {
2143     \tl_gput_right:Nx \g_@@_internal_code_after_tl
2144       { \@@_delimiter:nnn #1 { \int_use:N \c@jCol } \c_false_bool }
2145     \@@_error:nn { double-closing-delimiter } { #2 }
2146     \@@_patch_preamble:n #3
2147   }
2148 }

```

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.

```

2149 \cs_new_protected:Npn \@@_patch_preamble_x:n #1
2150 {
2151   \str_if_eq:nnTF { #1 } { [ ]
2152     { \@@_patch_preamble_x_i:w [ ]
2153       { \@@_patch_preamble_x_i:w [ ] #1 }
2154     }
2155   \cs_new_protected:Npn \@@_patch_preamble_x_i:w [ #1 ]
2156     { \@@_patch_preamble_x_ii:n { #1 } }

```

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

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

```

2157 \keys_define:nn { WithArrows / X-column }
2158 { unknown .code:n = \int_set:Nn \l_@@_weight_int { \l_keys_key_str } }

```

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

```

2159 \cs_new_protected:Npn \@@_patch_preamble_x_ii:n #1
2160 {

```

The possible values of `\l_@@_hpos_col_str` are j (for *justified* which is the initial value), l, c and r (when the user has used the corresponding key in the optional argument of the specifier X).

```

2161   \str_set:Nn \l_@@_hpos_col_str { j }

```

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

```
2162 \tl_set:Nn \l_@@_vpos_col_str { p }
```

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

```
2163 \int_zero_new:N \l_@@_weight_int
2164 \int_set:Nn \l_@@_weight_int { 1 }
2165 \keys_set_known:nnN { WithArrows / p-column } { #1 } \l_tmpa_tl
2166 \keys_set:nV { WithArrows / X-column } \l_tmpa_tl
2167 \int_compare:nNnT \l_@@_weight_int < 0
2168 {
2169   \exp_args:Nnx \@@_error:nn { negative~weight }
2170   { \int_use:N \l_@@_weight_int }
2171   \int_set:Nn \l_@@_weight_int { - \l_@@_weight_int }
2172 }
2173 \int_gadd:Nn \g_@@_total_X_weight_int \l_@@_weight_int
```

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

```
2174 \bool_if:NTF \l_@@_X_columns_aux_bool
2175 {
2176   \@@_patch_preamble_iv_iv:nn
2177   { \l_@@_weight_int \l_@@_X_columns_dim }
2178   { minipage }
2179 }
2180 {
2181   \tl_gput_right:Nn \g_@@_preamble_tl
2182   {
2183     > {
2184       \@@_cell_begin:w
2185       \bool_set_true:N \l_@@_X_column_bool
```

The following code will nullify the box of the cell.

```
2186 \tl_gput_right:Nn \g_@@_post_action_cell_tl
2187 { \hbox_set:Nn \l_@@_cell_box { } }
```

We put a `{minipage}` to give to the user the ability to put a command such as `\centering` in the `\RowStyle`.

```
2188 \begin { minipage } { 5 cm } \arraybackslash
2189 }
2190 c
2191 < {
2192   \end { minipage }
2193   \@@_cell_end:
2194 }
2195 }
2196 \int_gincr:N \c@jCol
2197 \@@_patch_preamble_xi:n
2198 }
2199 }
```

```
2200 \cs_new_protected:Npn \@@_patch_preamble_xii:n #1
2201 {
2202   \tl_gput_right:Nn \g_@@_preamble_tl
2203   { ! { \skip_horizontal:N 2\l_@@_radius_dim } }
```

The command `\@@_vdottedline:n` is protected, and, therefore, won't be expanded before writing on `\g_@@_internal_code_after_tl`.

```
2204 \tl_gput_right:Nx \g_@@_internal_code_after_tl
2205 { \@@_vdottedline:n { \int_use:N \c@jCol } }
2206 \@@_patch_preamble:n
2207 }
```

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.

```

2208 \cs_new_protected:Npn \@@_patch_preamble_xi:n #1
2209 {
2210   \str_if_eq:nnTF { #1 } { < }
2211     \@@_patch_preamble_xiii:n
2212     {
2213       \tl_if_eq:NnTF \l_@@_vlines_clist { all }
2214       {
2215         \tl_gput_right:Nn \g_@@_preamble_tl
2216           { ! { \skip_horizontal:N \arrayrulewidth } }
2217       }
2218       {
2219         \exp_args:NNx
2220         \clist_if_in:NnT \l_@@_vlines_clist { \@@_succ:n \c@jCol }
2221         {
2222           \tl_gput_right:Nn \g_@@_preamble_tl
2223             { ! { \skip_horizontal:N \arrayrulewidth } }
2224         }
2225       }
2226       \@@_patch_preamble:n { #1 }
2227     }
2228 }
2229 \cs_new_protected:Npn \@@_patch_preamble_xiii:n #1
2230 {
2231   \tl_gput_right:Nn \g_@@_preamble_tl { < { #1 } }
2232   \@@_patch_preamble_xi:n
2233 }

```

The redefinition of `\multicolumn`

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

```

2234 \cs_new:Npn \@@_multicolumn:nnn #1 #2 #3
2235 {

```

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

```

2236   \multispan { #1 }
2237   \begingroup
2238   \cs_set:Npn \@addamp { \if@firstamp \@firstampfalse \else \@preamerr 5 \fi }

```

You do the expansion of the (small) preamble with the tools of `array`.

```

2239   \@temptokena = { #2 }
2240   \@tempswattrue
2241   \@whilesw \if@tempswa \fi { \@tempswafalse \the \NC@list }

```

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

```

2242   \tl_gclear:N \g_@@_preamble_tl
2243   \exp_after:wN \@@_patch_m_preamble:n \the \@temptokena \q_stop

```

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

```

2244   \exp_args:NV \mkpream \g_@@_preamble_tl
2245   \addtopreamble \empty
2246   \endgroup

```

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

```

2247 \int_compare:nNnT { #1 } > 1
2248 {
2249   \seq_gput_left:Nx \g_@@_multicolumn_cells_seq
2250   { \int_use:N \c@iRow - \@@_succ:n \c@jCol }
2251   \seq_gput_left:Nn \g_@@_multicolumn_sizes_seq { #1 }
2252   \seq_gput_right:Nx \g_@@_pos_of_blocks_seq
2253   {
2254     {
2255       \int_compare:nNnTF \c@jCol = 0
2256       { \int_eval:n { \c@iRow + 1 } }
2257       { \int_use:N \c@iRow }
2258     } % modified 2022/01/10
2259     { \int_eval:n { \c@jCol + 1 } }
2260     {
2261       \int_compare:nNnTF \c@jCol = 0
2262       { \int_eval:n { \c@iRow + 1 } }
2263       { \int_use:N \c@iRow }
2264     } % modified 2022/01/10
2265     { \int_eval:n { \c@jCol + #1 } }
2266     { } % for the name of the block
2267   }
2268 }

```

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

```

2269 \cs_set:Npn \@sharp { #3 }
2270 \@arstrut
2271 \@preamble
2272 \null

```

We add some lines.

```

2273 \int_gadd:Nn \c@jCol { #1 - 1 }
2274 \int_compare:nNnT \c@jCol > \g_@@_col_total_int
2275 { \int_gset_eq:NN \g_@@_col_total_int \c@jCol }
2276 \ignorespaces
2277 }

```

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`.

```

2278 \cs_new_protected:Npn \@@_patch_m_preamble:n #1
2279 {
2280   \str_case:nnF { #1 }
2281   {
2282     c { \@@_patch_m_preamble_i:n #1 }
2283     l { \@@_patch_m_preamble_i:n #1 }
2284     r { \@@_patch_m_preamble_i:n #1 }
2285     > { \@@_patch_m_preamble_ii:nn #1 }
2286     ! { \@@_patch_m_preamble_ii:nn #1 }
2287     @ { \@@_patch_m_preamble_ii:nn #1 }
2288     | { \@@_patch_m_preamble_iii:n #1 }
2289     p { \@@_patch_m_preamble_iv:nnn t #1 }
2290     m { \@@_patch_m_preamble_iv:nnn c #1 }
2291     b { \@@_patch_m_preamble_iv:nnn b #1 }
2292     \@@_w: { \@@_patch_m_preamble_v:nnnn { } #1 }
2293     \@@_W: { \@@_patch_m_preamble_v:nnnn { \cs_set_eq:NN \hss \hfil } #1 }
2294     \q_stop { }
2295   }
2296   { \@@_fatal:nn { unknown~column~type } { #1 } }
2297 }

```

For c, l and r

```

2298 \cs_new_protected:Npn \@@_patch_m_preamble_i:n #1
2299 {
2300   \tl_gput_right:Nn \g_@@_preamble_tl
2301   {
2302     > { \@@_cell_begin:w \str_set:Nn \l_@@_hpos_cell_str { #1 } }
2303     #1
2304     < \@@_cell_end:
2305   }

```

We test for the presence of a <.

```

2306   \@@_patch_m_preamble_x:n
2307 }

```

For >, ! and @

```

2308 \cs_new_protected:Npn \@@_patch_m_preamble_ii:nn #1 #2
2309 {
2310   \tl_gput_right:Nn \g_@@_preamble_tl { #1 { #2 } }
2311   \@@_patch_m_preamble:n
2312 }

```

For |

```

2313 \cs_new_protected:Npn \@@_patch_m_preamble_iii:n #1
2314 {
2315   \tl_gput_right:Nn \g_@@_preamble_tl { #1 }
2316   \@@_patch_m_preamble:n
2317 }

```

For p, m and b

```

2318 \cs_new_protected:Npn \@@_patch_m_preamble_iv:nnn #1 #2 #3
2319 {
2320   \tl_gput_right:Nn \g_@@_preamble_tl
2321   {
2322     > {
2323       \@@_cell_begin:w
2324       \begin { minipage } [ #1 ] { \dim_eval:n { #3 } }
2325       \mode_leave_vertical:
2326       \arraybackslash
2327       \vrule height \box_ht:N \@arstrutbox depth 0 pt width 0 pt
2328     }
2329     c
2330     < {
2331       \vrule height 0 pt depth \box_dp:N \@arstrutbox width 0 pt
2332       \end { minipage }
2333       \@@_cell_end:
2334     }
2335   }

```

We test for the presence of a <.

```

2336   \@@_patch_m_preamble_x:n
2337 }

```

For w and W

```

2338 \cs_new_protected:Npn \@@_patch_m_preamble_v:nnnn #1 #2 #3 #4
2339 {
2340   \tl_gput_right:Nn \g_@@_preamble_tl
2341   {
2342     > {
2343       \hbox_set:Nw \l_@@_cell_box
2344       \@@_cell_begin:w
2345       \str_set:Nn \l_@@_hpos_cell_str { #3 }
2346     }
2347     c
2348     < {

```

```

2349         \@@_cell_end:
2350         #1
2351         \hbox_set_end:
2352         \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
2353         \@@_adjust_size_box:
2354         \makebox [ #4 ] [ #3 ] { \box_use_drop:N \l_@@_cell_box }
2355     }
2356 }

```

We test for the presence of a <.

```

2357     \@@_patch_m_preamble_x:n
2358 }

```

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.

```

2359 \cs_new_protected:Npn \@@_patch_m_preamble_x:n #1
2360 {
2361     \str_if_eq:nnTF { #1 } { < }
2362     \@@_patch_m_preamble_ix:n
2363     {
2364         \tl_if_eq:NnTF \l_@@_vlines_clist { all }
2365         {
2366             \tl_gput_right:Nn \g_@@_preamble_tl
2367             { ! { \skip_horizontal:N \arrayrulewidth } }
2368         }
2369         {
2370             \exp_args:NNx
2371             \clist_if_in:NnT \l_@@_vlines_clist { \@@_succ:n \c@jCol }
2372             {
2373                 \tl_gput_right:Nn \g_@@_preamble_tl
2374                 { ! { \skip_horizontal:N \arrayrulewidth } }
2375             }
2376         }
2377         \@@_patch_m_preamble:n { #1 }
2378     }
2379 }
2380 \cs_new_protected:Npn \@@_patch_m_preamble_ix:n #1
2381 {
2382     \tl_gput_right:Nn \g_@@_preamble_tl { < { #1 } }
2383     \@@_patch_m_preamble_x:n
2384 }

```

The command `\@@_put_box_in_flow:` puts the box `\l_tmpa_box` (which contains the array) in the flow. It is used for the environments with delimiters. First, we have to modify the height and the depth to take back into account the potential exterior rows (the total height of the first row has been computed in `\l_tmpa_dim` and the total height of the potential last row in `\l_tmpb_dim`).

```

2385 \cs_new_protected:Npn \@@_put_box_in_flow:
2386 {
2387     \box_set_ht:Nn \l_tmpa_box { \box_ht:N \l_tmpa_box + \l_tmpa_dim }
2388     \box_set_dp:Nn \l_tmpa_box { \box_dp:N \l_tmpa_box + \l_tmpb_dim }
2389     \tl_if_eq:NnTF \l_@@_baseline_tl { c }
2390     { \box_use_drop:N \l_tmpa_box }
2391     \@@_put_box_in_flow_i:
2392 }

```

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).

```

2393 \cs_new_protected:Npn \@@_put_box_in_flow_i:
2394 {
2395     \pgfpicture
2396     \@@_qpoint:n { row - 1 }
2397     \dim_gset_eq:NN \g_tmpa_dim \pgf@y

```

```

2398 \@@_qpoint:n { row - \@@_succ:n \c@iRow }
2399 \dim_gadd:Nn \g_tmpa_dim \pgf@y
2400 \dim_gset:Nn \g_tmpa_dim { 0.5 \g_tmpa_dim }

```

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

```

2401 \str_if_in:NnTF \l_@@_baseline_tl { line- }
2402 {
2403   \int_set:Nn \l_tmpa_int
2404   {
2405     \str_range:Nnn
2406       \l_@@_baseline_tl
2407       6
2408     { \tl_count:V \l_@@_baseline_tl }
2409   }
2410   \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
2411 }
2412 {
2413   \str_case:VnF \l_@@_baseline_tl
2414   {
2415     { t } { \int_set:Nn \l_tmpa_int 1 }
2416     { b } { \int_set_eq:NN \l_tmpa_int \c@iRow }
2417   }
2418   { \int_set:Nn \l_tmpa_int \l_@@_baseline_tl }
2419   \bool_lazy_or:nnT
2420     { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
2421     { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
2422   {
2423     \@@_error:n { bad~value~for~baseline }
2424     \int_set:Nn \l_tmpa_int 1
2425   }
2426   \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }

```

We take into account the position of the mathematical axis.

```

2427 \dim_gsub:Nn \g_tmpa_dim { \fontdimen22 \textfont2 }
2428 }
2429 \dim_gsub:Nn \g_tmpa_dim \pgf@y

```

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

```

2430 \endpgfpicture
2431 \box_move_up:nn \g_tmpa_dim { \box_use_drop:N \l_tmpa_box }
2432 \box_use_drop:N \l_tmpa_box
2433 }

```

The following command is *always* used by `{NiceArrayWithDelims}` (even if, in fact, there is no tabular notes: in fact, it's not possible to know whether there is tabular notes or not before the composition of the blocks).

```

2434 \cs_new_protected:Npn \@@_use_arraybox_with_notes_c:
2435 {

```

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.

```

2436 \bool_lazy_and:nnT \l_@@_Matrix_bool \l_@@_NiceArray_bool
2437 {
2438   \box_set_wd:Nn \l_@@_the_array_box
2439   { \box_wd:N \l_@@_the_array_box - \arraycolsep }
2440 }

```

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

```

2441 \begin { minipage } [ t ] { \box_wd:N \l_@@_the_array_box }

```

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

```

2442   \hbox
2443   {
2444     \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.

```

2445     \@@_create_extra_nodes:
2446     \seq_if_empty:NF \g_@@_blocks_seq \@@_draw_blocks:
2447   }
2448   \bool_lazy_or:nnT
2449   { \int_compare_p:nNn \c@tabularnote > 0 }
2450   { ! \tl_if_empty_p:V \l_@@_tabularnote_tl }
2451   \@@_insert_tabularnotes:
2452   \end { minipage }
2453 }
2454 \cs_new_protected:Npn \@@_insert_tabularnotes:
2455 {
2456   \skip_vertical:N 0.65ex

```

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

```

2457   \group_begin:
2458   \l_@@_notes_code_before_tl
2459   \tl_if_empty:NF \l_@@_tabularnote_tl { \l_@@_tabularnote_tl \par }

```

We compose the tabular notes with a list of `enumitem`. The `\strut` and the `\unskip` are designed to give the ability to put a `\bottomrule` at the end of the notes with a good vertical space.

```

2460   \int_compare:nNnT \c@tabularnote > 0
2461   {
2462     \bool_if:NTF \l_@@_notes_para_bool
2463     {
2464       \begin { tabularnotes* }
2465       \seq_map_inline:Nn \g_@@_tabularnotes_seq { \item ##1 } \strut
2466       \end { tabularnotes* }

```

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

```

2467     \par
2468   }
2469   {
2470     \tabularnotes
2471     \seq_map_inline:Nn \g_@@_tabularnotes_seq { \item ##1 } \strut
2472     \endtabularnotes
2473   }
2474 }
2475 \unskip
2476 \group_end:
2477 \bool_if:NT \l_@@_notes_bottomrule_bool
2478 {
2479   \bool_if:NTF \c_@@_booktabs_loaded_bool
2480   {

```

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

```

2481     \skip_vertical:N \aboverulesep
\CT@arc@ is the specification of color defined by colortbl but you use it even if colortbl is not loaded.
2482     { \CT@arc@ \hrule height \heavyrulewidth }
2483   }
2484   { \@_error:n { bottomrule~without~booktabs } }
2485 }
2486 \l_@@_notes_code_after_tl

```



```

2487 \seq_gclear:N \g_@@_tabularnotes_seq
2488 \int_gzero:N \c@tabularnote
2489 }

```

The case of baseline equal to b. Remember that, when the key b is used, the {array} (of array) is constructed with the option t (and not b). Now, we do the translation to take into account the option b.

```

2490 \cs_new_protected:Npn \@@_use_arraybox_with_notes_b:
2491 {
2492   \pgfpicture
2493     \@@_qpoint:n { row - 1 }
2494     \dim_gset_eq:NN \g_tmpa_dim \pgf@y
2495     \@@_qpoint:n { row - \int_use:N \c@iRow - base }
2496     \dim_gsub:Nn \g_tmpa_dim \pgf@y
2497   \endpgfpicture
2498   \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
2499   \int_compare:nNnT \l_@@_first_row_int = 0
2500   {
2501     \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
2502     \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
2503   }
2504   \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
2505 }

```

Now, the general case.

```

2506 \cs_new_protected:Npn \@@_use_arraybox_with_notes:
2507 {

```

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

```

2508   \tl_if_eq:NnT \l_@@_baseline_tl { t }
2509   { \tl_set:Nn \l_@@_baseline_tl { 1 } }

```

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

```

2510   \pgfpicture
2511   \@@_qpoint:n { row - 1 }
2512   \dim_gset_eq:NN \g_tmpa_dim \pgf@y
2513   \str_if_in:NnTF \l_@@_baseline_tl { line- }
2514   {
2515     \int_set:Nn \l_tmpa_int
2516     {
2517       \str_range:Nnn
2518         \l_@@_baseline_tl
2519         6
2520       { \tl_count:V \l_@@_baseline_tl }
2521     }
2522     \@@_qpoint:n { row - \int_use:N \l_tmpa_int }
2523   }
2524   {
2525     \int_set:Nn \l_tmpa_int \l_@@_baseline_tl
2526     \bool_lazy_or:nnT
2527     { \int_compare_p:nNn \l_tmpa_int < \l_@@_first_row_int }
2528     { \int_compare_p:nNn \l_tmpa_int > \g_@@_row_total_int }
2529     {
2530       \@@_error:n { bad~value~for~baseline }
2531       \int_set:Nn \l_tmpa_int 1
2532     }
2533     \@@_qpoint:n { row - \int_use:N \l_tmpa_int - base }
2534   }
2535   \dim_gsub:Nn \g_tmpa_dim \pgf@y
2536   \endpgfpicture
2537   \dim_gadd:Nn \g_tmpa_dim \arrayrulewidth
2538   \int_compare:nNnT \l_@@_first_row_int = 0
2539   {

```

```

2540     \dim_gadd:Nn \g_tmpa_dim \g_@@_ht_row_zero_dim
2541     \dim_gadd:Nn \g_tmpa_dim \g_@@_dp_row_zero_dim
2542   }
2543   \box_move_up:nn \g_tmpa_dim { \hbox { \@@_use_arraybox_with_notes_c: } }
2544 }

```

The command `\@@_put_box_in_flow_bis:` is used when the option `delimiters/max-width` is used because, in this case, we have to adjust the widths of the delimiters. The arguments `#1` and `#2` are the delimiters specified by the user.

```

2545 \cs_new_protected:Npn \@@_put_box_in_flow_bis:nn #1 #2
2546 {

```

We will compute the real width of both delimiters used.

```

2547   \dim_zero_new:N \l_@@_real_left_delim_dim
2548   \dim_zero_new:N \l_@@_real_right_delim_dim
2549   \hbox_set:Nn \l_tmpb_box
2550   {
2551     \c_math_toggle_token
2552     \left #1
2553     \vcenter
2554     {
2555       \vbox_to_ht:nn
2556       { \box_ht_plus_dp:N \l_tmpa_box }
2557       { }
2558     }
2559     \right .
2560     \c_math_toggle_token
2561   }
2562   \dim_set:Nn \l_@@_real_left_delim_dim
2563   { \box_wd:N \l_tmpb_box - \nulldelimiterspace }
2564   \hbox_set:Nn \l_tmpb_box
2565   {
2566     \c_math_toggle_token
2567     \left .
2568     \vbox_to_ht:nn
2569     { \box_ht_plus_dp:N \l_tmpa_box }
2570     { }
2571     \right #2
2572     \c_math_toggle_token
2573   }
2574   \dim_set:Nn \l_@@_real_right_delim_dim
2575   { \box_wd:N \l_tmpb_box - \nulldelimiterspace }

```

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

```

2576   \skip_horizontal:N \l_@@_left_delim_dim
2577   \skip_horizontal:N -\l_@@_real_left_delim_dim
2578   \@@_put_box_in_flow:
2579   \skip_horizontal:N \l_@@_right_delim_dim
2580   \skip_horizontal:N -\l_@@_real_right_delim_dim
2581 }

```

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

```

2582 \NewDocumentEnvironment { @@-normal-syntax } { }

```

First, we test whether the environment is empty. If it is empty, we raise a fatal error (it's only a security). In order to detect whether it is empty, we test whether the next token is `\end` and, if it's the case, we test if this is the end of the environment (if it is not, an standard error will be raised by LaTeX for incorrect nested environments).

```

2583 {
2584   \peek_meaning_ignore_spaces:NTF \end \@@_analyze_end:Nn

```

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

```

2585     { \exp_args:NV \@@_array: \g_@@_preamble_tl }
2586   }
2587   {
2588     \@@_create_col_nodes:
2589     \endarray
2590   }

```

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

```

2591 \NewDocumentEnvironment { @@-light-syntax } { b }
2592   {

```

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`.

```

2593   \tl_if_empty:nT { #1 } { \@@_fatal:n { empty-environment } }
2594   \tl_map_inline:nn { #1 }
2595     {
2596       \str_if_eq:nnT { ##1 } { & }
2597       { \@@_fatal:n { ampersand-in-light-syntax } }
2598       \str_if_eq:nnT { ##1 } { \ }
2599       { \@@_fatal:n { double-backslash-in-light-syntax } }
2600     }

```

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

```

2601   \@@_light_syntax_i #1 \CodeAfter \q_stop
2602   }

```

Now, the second part of the environment. It is empty. That's not surprising because we have caught the whole body of the environment with the specifier `b` provided by `xparse`.

```

2603   { }

2604 \cs_new_protected:Npn \@@_light_syntax_i #1\CodeAfter #2\q_stop
2605   {
2606     \tl_gput_right:Nn \g_nicematrix_code_after_tl { #2 }

```

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

```

2607   \seq_gclear_new:N \g_@@_rows_seq
2608   \tl_set_rescan:Nno \l_@@_end_of_row_tl { } \l_@@_end_of_row_tl
2609   \exp_args:NNV \seq_gset_split:Nnn \g_@@_rows_seq \l_@@_end_of_row_tl { #1 }

```

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 know 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.

```

2610   \int_compare:nNnT \l_@@_last_row_int = { -1 }
2611     { \int_set:Nn \l_@@_last_row_int { \seq_count:N \g_@@_rows_seq } }

```

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

```

2612   \exp_args:NV \@@_array: \g_@@_preamble_tl

```

We need a global affectation because, when executing `\l_tmpa_tl`, we will exit the first cell of the array.

```

2613   \seq_gpop_left:NN \g_@@_rows_seq \l_tmpa_tl
2614   \exp_args:NV \@@_line_with_light_syntax_i:n \l_tmpa_tl
2615   \seq_map_function:NN \g_@@_rows_seq \@@_line_with_light_syntax:n
2616   \@@_create_col_nodes:
2617   \endarray
2618   }

```

```

2619 \cs_new_protected:Npn \@@_line_with_light_syntax:n #1
2620 { \tl_if_empty:nF { #1 } { \ \ \@@_line_with_light_syntax_i:n { #1 } } }
2621 \cs_new_protected:Npn \@@_line_with_light_syntax_i:n #1
2622 {
2623   \seq_gclear_new:N \g_@@_cells_seq
2624   \seq_gset_split:Nnn \g_@@_cells_seq { ~ } { #1 }
2625   \seq_gpop_left:NN \g_@@_cells_seq \l_tmpa_tl
2626   \l_tmpa_tl
2627   \seq_map_inline:Nn \g_@@_cells_seq { & ##1 }
2628 }

```

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).

```

2629 \cs_new_protected:Npn \@@_analyze_end:Nn #1 #2
2630 {
2631   \str_if_eq:VnT \g_@@_name_env_str { #2 }
2632   { \@@_fatal:n { empty~environment } }

```

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

```

2633   \end { #2 }
2634 }

```

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

```

2635 \cs_new:Npn \@@_create_col_nodes:
2636 {
2637   \crcr
2638   \int_compare:nNnT \l_@@_first_col_int = 0
2639   {
2640     \omit
2641     \hbox_overlap_left:n
2642     {
2643       \bool_if:NT \l_@@_code_before_bool
2644       { \pgfsys@markposition { \@@_env: - col - 0 } }
2645       \pgfpicture
2646       \pgfrememberpicturerepositiononpagetrue
2647       \pgfcoordinate { \@@_env: - col - 0 } \pgfpointorigin
2648       \str_if_empty:NF \l_@@_name_str
2649       { \pgfnodealias { \l_@@_name_str - col - 0 } { \@@_env: - col - 0 } }
2650       \endpgfpicture
2651       \skip_horizontal:N 2\col@sep
2652       \skip_horizontal:N \g_@@_width_first_col_dim
2653     }
2654     &
2655   }
2656   \omit

```

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

```

2657   \bool_gset_true:N \g_@@_row_of_col_done_bool

```

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

```

2658   \int_compare:nNnTF \l_@@_first_col_int = 0
2659   {
2660     \bool_if:NT \l_@@_code_before_bool
2661     {
2662       \hbox
2663       {
2664         \skip_horizontal:N -0.5\arrayrulewidth
2665         \pgfsys@markposition { \@@_env: - col - 1 }

```

```

2666         \skip_horizontal:N 0.5\arrayrulewidth
2667     }
2668 }
2669 \pgfpicture
2670 \pgfrememberpicturepositiononpagetrue
2671 \pgfcoordinate { \@@_env: - col - 1 }
2672 { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
2673 \str_if_empty:NF \l_@@_name_str
2674 { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
2675 \endpgfpicture
2676 }
2677 {
2678     \bool_if:NT \l_@@_code_before_bool
2679     {
2680         \hbox
2681         {
2682             \skip_horizontal:N 0.5\arrayrulewidth
2683             \pgfsys@markposition { \@@_env: - col - 1 }
2684             \skip_horizontal:N -0.5\arrayrulewidth
2685         }
2686     }
2687     \pgfpicture
2688     \pgfrememberpicturepositiononpagetrue
2689     \pgfcoordinate { \@@_env: - col - 1 }
2690     { \pgfpoint { 0.5 \arrayrulewidth } \c_zero_dim }
2691     \str_if_empty:NF \l_@@_name_str
2692     { \pgfnodealias { \l_@@_name_str - col - 1 } { \@@_env: - col - 1 } }
2693     \endpgfpicture
2694 }

```

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

We give a default value for `\g_tmpa_skip` (0 pt plus 1 fill) but it will just after be erased by a fixed value in the concerned cases.

```

2695     \skip_gset:Nn \g_tmpa_skip { 0 pt-plus 1 fill }
2696     \bool_if:NF \l_@@_auto_columns_width_bool
2697     { \dim_compare:nNnT \l_@@_columns_width_dim > \c_zero_dim }
2698     {
2699         \bool_lazy_and:nnTF
2700         \l_@@_auto_columns_width_bool
2701         { \bool_not_p:n \l_@@_block_auto_columns_width_bool }
2702         { \skip_gset_eq:NN \g_tmpa_skip \g_@@_max_cell_width_dim }
2703         { \skip_gset_eq:NN \g_tmpa_skip \l_@@_columns_width_dim }
2704         \skip_gadd:Nn \g_tmpa_skip { 2 \col@sep }
2705     }
2706     \skip_horizontal:N \g_tmpa_skip
2707     \hbox
2708     {
2709         \bool_if:NT \l_@@_code_before_bool
2710         {
2711             \hbox
2712             {
2713                 \skip_horizontal:N -0.5\arrayrulewidth
2714                 \pgfsys@markposition { \@@_env: - col - 2 }
2715                 \skip_horizontal:N 0.5\arrayrulewidth
2716             }
2717         }
2718         \pgfpicture
2719         \pgfrememberpicturepositiononpagetrue
2720         \pgfcoordinate { \@@_env: - col - 2 }
2721         { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }

```

```

2722 \str_if_empty:NF \l_@@_name_str
2723 { \pgfnodealias { \l_@@_name_str - col - 2 } { \@@_env: - col - 2 } }
2724 \endpgfpicture
2725 }

```

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.

```

2726 \int_gset:Nn \g_tmpa_int 1
2727 \bool_if:NTF \g_@@_last_col_found_bool
2728 { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 3 } 0 } }
2729 { \prg_replicate:nn { \int_max:nn { \g_@@_col_total_int - 2 } 0 } }
2730 {
2731   &
2732   \omit
2733   \int_gincr:N \g_tmpa_int

```

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

```

2734 \skip_horizontal:N \g_tmpa_skip
2735 \bool_if:NT \l_@@_code_before_bool
2736 {
2737   \hbox
2738   {
2739     \skip_horizontal:N -0.5\arrayrulewidth
2740     \pgfsys@markposition { \@@_env: - col - \@@_succ:n \g_tmpa_int }
2741     \skip_horizontal:N 0.5\arrayrulewidth
2742   }
2743 }

```

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

```

2744 \pgfpicture
2745 \pgfrememberpicturepositiononpagetrue
2746 \pgfcoordinate { \@@_env: - col - \@@_succ:n \g_tmpa_int }
2747 { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
2748 \str_if_empty:NF \l_@@_name_str
2749 {
2750   \pgfnodealias
2751   { \l_@@_name_str - col - \@@_succ:n \g_tmpa_int }
2752   { \@@_env: - col - \@@_succ:n \g_tmpa_int }
2753 }
2754 \endpgfpicture
2755 }

```

```

2756 &
2757 \omit

```

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

```

2758 \int_compare:nNnT \g_@@_col_total_int = 1
2759 { \skip_gset:Nn \g_tmpa_skip { 0 pt~plus 1 fill } }
2760 \skip_horizontal:N \g_tmpa_skip
2761 \int_gincr:N \g_tmpa_int
2762 \bool_lazy_all:nT
2763 {
2764   \l_@@_NiceArray_bool
2765   { \bool_not_p:n \l_@@_NiceTabular_bool }
2766   { \clist_if_empty_p:N \l_@@_vlines_clist }
2767   { \bool_not_p:n \l_@@_exterior_arraycolsep_bool }
2768   { ! \l_@@_bar_at_end_of_pream_bool }
2769 }
2770 { \skip_horizontal:N -\col@sep }
2771 \bool_if:NT \l_@@_code_before_bool
2772 {
2773   \hbox
2774   {
2775     \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.

```

2776         \bool_lazy_and:nnT \l_@@_Matrix_bool \l_@@_NiceArray_bool
2777         { \skip_horizontal:N -\arraycolsep }
2778         \pgfsys@markposition { \@@_env: - col - \@@_succ:n \g_tmpa_int }
2779         \skip_horizontal:N 0.5\arrayrulewidth
2780         \bool_lazy_and:nnT \l_@@_Matrix_bool \l_@@_NiceArray_bool
2781         { \skip_horizontal:N \arraycolsep }
2782     }
2783 }
2784 \pgfpicture
2785 \pgfrememberpicturepositiononpagetrue
2786 \pgfcoordinate { \@@_env: - col - \@@_succ:n \g_tmpa_int }
2787 {
2788     \bool_lazy_and:nnTF \l_@@_Matrix_bool \l_@@_NiceArray_bool
2789     {
2790         \pgfpoint
2791         { - 0.5 \arrayrulewidth - \arraycolsep }
2792         \c_zero_dim
2793     }
2794     { \pgfpoint { - 0.5 \arrayrulewidth } \c_zero_dim }
2795 }
2796 \str_if_empty:NF \l_@@_name_str
2797 {
2798     \pgfnodealias
2799     { \l_@@_name_str - col - \@@_succ:n \g_tmpa_int }
2800     { \@@_env: - col - \@@_succ:n \g_tmpa_int }
2801 }
2802 \endpgfpicture

2803 \bool_if:NT \g_@@_last_col_found_bool
2804 {
2805     \hbox_overlap_right:n
2806     {
2807         \skip_horizontal:N \g_@@_width_last_col_dim
2808         \bool_if:NT \l_@@_code_before_bool
2809         {
2810             \pgfsys@markposition
2811             { \@@_env: - col - \@@_succ:n \g_@@_col_total_int }
2812         }
2813         \pgfpicture
2814         \pgfrememberpicturepositiononpagetrue
2815         \pgfcoordinate { \@@_env: - col - \@@_succ:n \g_@@_col_total_int }
2816         \pgfpointorigin
2817         \str_if_empty:NF \l_@@_name_str
2818         {
2819             \pgfnodealias
2820             { \l_@@_name_str - col - \@@_succ:n \g_@@_col_total_int }
2821             { \@@_env: - col - \@@_succ:n \g_@@_col_total_int }
2822         }
2823         \endpgfpicture
2824     }
2825 }
2826 \cr
2827 }
```

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

```

2828 \tl_const:Nn \c_@@_preamble_first_col_tl
2829 {
2830     >
2831     {
```

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).

```
2832 \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
2833 \bool_gset_true:N \g_@@_after_col_zero_bool
2834 \@@_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.

```
2835 \hbox_set:Nw \l_@@_cell_box
2836 \@@_math_toggle_token:
2837 \bool_if:NT \l_@@_small_bool \scriptstyle
```

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

```
2838 \bool_lazy_and:nnT
2839 { \int_compare_p:nNn \c@iRow > 0 }
2840 {
2841   \bool_lazy_or_p:nn
2842   { \int_compare_p:nNn \l_@@_last_row_int < 0 }
2843   { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
2844 }
2845 {
2846   \l_@@_code_for_first_col_tl
2847   \xglobal \colorlet { nicematrix-first-col } { . }
2848 }
2849 }
```

Be careful: despite this letter `l` the cells of the “first column” are composed in a `R` manner since they are composed in a `\hbox_overlap_left:n`.

```
2850 l
2851 <
2852 {
2853   \@@_math_toggle_token:
2854   \hbox_set_end:
2855   \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
2856   \@@_adjust_size_box:
2857   \@@_update_for_first_and_last_row:
```

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

```
2858 \dim_gset:Nn \g_@@_width_first_col_dim
2859 { \dim_max:nn \g_@@_width_first_col_dim { \box_wd:N \l_@@_cell_box } }
```

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

```
2860 \hbox_overlap_left:n
2861 {
2862   \dim_compare:nNnTF { \box_wd:N \l_@@_cell_box } > \c_zero_dim
2863   \@@_node_for_cell:
2864   { \box_use_drop:N \l_@@_cell_box }
2865   \skip_horizontal:N \l_@@_left_delim_dim
2866   \skip_horizontal:N \l_@@_left_margin_dim
2867   \skip_horizontal:N \l_@@_extra_left_margin_dim
2868 }
2869 \bool_gset_false:N \g_@@_empty_cell_bool
2870 \skip_horizontal:N -2\col@sep
2871 }
2872 }
```

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

```
2873 \tl_const:Nn \c_@@_preamble_last_col_tl
2874 {
2875   >
2876   {
```


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).

```
2877 \cs_set_eq:NN \CodeAfter \@@_CodeAfter_i:
```

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

```
2878 \bool_gset_true:N \g_@@_last_col_found_bool
2879 \int_gincr:N \c@jCol
2880 \int_gset_eq:NN \g_@@_col_total_int \c@jCol
```

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

```
2881 \hbox_set:Nw \l_@@_cell_box
2882 \@@_math_toggle_token:
2883 \bool_if:NT \l_@@_small_bool \scriptstyle
```

We insert `\l_@@_code_for_last_col_tl...` but we don’t insert it in the potential “first row” and in the potential “last row”.

```
2884 \int_compare:nNnT \c@iRow > 0
2885 {
2886   \bool_lazy_or:nnT
2887   { \int_compare_p:nNn \l_@@_last_row_int < 0 }
2888   { \int_compare_p:nNn \c@iRow < \l_@@_last_row_int }
2889   {
2890     \l_@@_code_for_last_col_tl
2891     \xglobal \colorlet { nicematrix-last-col } { . }
2892   }
2893 }
2894 }
2895 1
2896 <
2897 {
2898   \@@_math_toggle_token:
2899   \hbox_set_end:
2900   \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:
2901   \@@_adjust_size_box:
2902   \@@_update_for_first_and_last_row:
```

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

```
2903 \dim_gset:Nn \g_@@_width_last_col_dim
2904 { \dim_max:nn \g_@@_width_last_col_dim { \box_wd:N \l_@@_cell_box } }
2905 \skip_horizontal:N -2\col@sep
```

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

```
2906 \hbox_overlap_right:n
2907 {
2908   \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } > \c_zero_dim
2909   {
2910     \skip_horizontal:N \l_@@_right_delim_dim
2911     \skip_horizontal:N \l_@@_right_margin_dim
2912     \skip_horizontal:N \l_@@_extra_right_margin_dim
2913     \@@_node_for_cell:
2914   }
2915 }
2916 \bool_gset_false:N \g_@@_empty_cell_bool
2917 }
2918 }
```

The environment `{NiceArray}` is constructed upon the environment `{NiceArrayWithDelims}` but, in fact, there is a flag `\l_@@_NiceArray_bool`. In `{NiceArrayWithDelims}`, some special code will be executed if this flag is raised.

```
2919 \NewDocumentEnvironment { NiceArray } { }
2920 {
```

```

2921 \bool_set_true:N \l_@@_NiceArray_bool
2922 \str_if_empty:NT \g_@@_name_env_str
2923 { \str_gset:Nn \g_@@_name_env_str { NiceArray } }

```

We put . and . for the delimiters but, in fact, that doesn't matter because these arguments won't be used in {NiceArrayWithDelims} (because the flag \l_@@_NiceArray_bool is raised).

```

2924 \NiceArrayWithDelims . .
2925 }
2926 { \endNiceArrayWithDelims }

```

We create the variants of the environment {NiceArrayWithDelims}.

```

2927 \cs_new_protected:Npn \@@_def_env:nnn #1 #2 #3
2928 {
2929   \NewDocumentEnvironment { #1 NiceArray } { }
2930   {
2931     \str_if_empty:NT \g_@@_name_env_str
2932     { \str_gset:Nn \g_@@_name_env_str { #1 NiceArray } }
2933     \@@_test_if_math_mode:
2934     \NiceArrayWithDelims #2 #3
2935   }
2936   { \endNiceArrayWithDelims }
2937 }
2938 \@@_def_env:nnn p ( )
2939 \@@_def_env:nnn b [ ]
2940 \@@_def_env:nnn B \{ \}
2941 \@@_def_env:nnn v | |
2942 \@@_def_env:nnn V \| \|

```

The environment {NiceMatrix} and its variants

```

2943 \cs_new_protected:Npn \@@_begin_of_NiceMatrix:nn #1 #2
2944 {
2945   \bool_set_true:N \l_@@_Matrix_bool
2946   \use:c { #1 NiceArray }
2947   {
2948     *
2949     {
2950       \int_compare:nNnTF \l_@@_last_col_int < 0
2951       \c@MaxMatrixCols
2952       { \@@_pred:n \l_@@_last_col_int }
2953     }
2954     { > \@@_cell_begin:w #2 < \@@_cell_end: }
2955   }
2956 }
2957 \clist_map_inline:nn { { } , p , b , B , v , V }
2958 {
2959   \NewDocumentEnvironment { #1 NiceMatrix } { ! O { } }
2960   {
2961     \str_gset:Nn \g_@@_name_env_str { #1 NiceMatrix }
2962     \tl_set:Nn \l_@@_type_of_col_tl c
2963     \keys_set:nn { NiceMatrix / NiceMatrix } { ##1 }
2964     \exp_args:Nne \@@_begin_of_NiceMatrix:nn { #1 } \l_@@_type_of_col_tl
2965   }
2966   { \use:c { end #1 NiceArray } }
2967 }

```

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

```

2968 \cs_new_protected:Npn \@@_NotEmpty:
2969 { \bool_gset_true:N \g_@@_not_empty_cell_bool }

```

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

```
2970 \NewDocumentEnvironment { NiceTabular } { 0 { } m ! 0 { } }
2971 {
```

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`.

```
2972   \dim_compare:nNnT \l_@@_width_dim = \c_zero_dim
2973     { \dim_set_eq:NN \l_@@_width_dim \linewidth }
2974   \str_gset:Nn \g_@@_name_env_str { NiceTabular }
2975   \keys_set:nn { NiceMatrix / NiceTabular } { #1 , #3 }
2976   \bool_set_true:N \l_@@_NiceTabular_bool
2977   \NiceArray { #2 }
2978 }
2979 { \endNiceArray }
```

```
2980 \cs_set_protected:Npn \@@_newcolumnntype #1
2981 {
2982   \cs_if_free:cT { NC @ find @ #1 }
2983     { \NC@list \expandafter { \the \NC@list \NC@do #1 } }
2984   \cs_set:cpn {NC @ find @ #1 } ##1 #1 { \NC@ { ##1 } }
2985   \peek_meaning:NTF [
2986     { \newcol@ #1 }
2987     { \newcol@ #1 [ 0 ] }
2988   }
```

```
2989 \NewDocumentEnvironment { NiceTabularX } { m 0 { } m ! 0 { } }
2990 {
```

The following code prevents the expansion of the ‘X’ columns with the definition of that columns in `tabularx` (this would result in an error in `{NiceTabularX}`).

```
2991   \bool_if:NT \c_@@_tabularx_loaded_bool
2992     { \newcolumnntype { X } { \@@_X } }
2993   \str_gset:Nn \g_@@_name_env_str { NiceTabularX }
2994   \dim_zero_new:N \l_@@_width_dim
2995   \dim_set:Nn \l_@@_width_dim { #1 }
2996   \keys_set:nn { NiceMatrix / NiceTabular } { #2 , #4 }
2997   \bool_set_true:N \l_@@_NiceTabular_bool
2998   \NiceArray { #3 }
2999 }
3000 { \endNiceArray }
```

```
3001 \NewDocumentEnvironment { NiceTabular* } { m 0 { } m ! 0 { } }
3002 {
3003   \str_gset:Nn \g_@@_name_env_str { NiceTabular* }
3004   \dim_set:Nn \l_@@_tabular_width_dim { #1 }
3005   \keys_set:nn { NiceMatrix / NiceTabular } { #2 , #4 }
3006   \bool_set_true:N \l_@@_NiceTabular_bool
3007   \NiceArray { #3 }
3008 }
3009 { \endNiceArray }
```

After the construction of the array

```
3010 \cs_new_protected:Npn \@@_after_array:
3011 {
3012   \group_begin:
```

When the option `last-col` is used in the environments with explicit preambles (like `{NiceArray}`, `{pNiceArray}`, etc.) a special type of column is used at the end of the preamble in order to compose the cells in an overlapping position (with `\hbox_overlap_right:n`) but (if `last-col` has been used), we don’t have the number of that last column. However, we have to know that number for the

color of the potential `\Vdots` drawn in that last column. That's why we fix the correct value of `\l_@@_last_col_int` in that case.

```
3013 \bool_if:NT \g_@@_last_col_found_bool
3014 { \int_set_eq:NN \l_@@_last_col_int \g_@@_col_total_int }
```

If we are in an environment without preamble (like `{NiceMatrix}` or `{pNiceMatrix}`) and if the option `last-col` has been used without value we also fix the real value of `\l_@@_last_col_int`.

```
3015 \bool_if:NT \l_@@_last_col_without_value_bool
3016 { \int_set_eq:NN \l_@@_last_col_int \g_@@_col_total_int }
```

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

```
3017 \bool_if:NT \l_@@_last_row_without_value_bool
3018 { \int_set_eq:NN \l_@@_last_row_int \g_@@_row_total_int }

3019 \tl_gput_right:Nx \g_@@_aux_tl
3020 {
3021   \seq_gset_from_clist:Nn \exp_not:N \c_@@_size_seq
3022   {
3023     \int_use:N \l_@@_first_row_int ,
3024     \int_use:N \c@iRow ,
3025     \int_use:N \g_@@_row_total_int ,
3026     \int_use:N \l_@@_first_col_int ,
3027     \int_use:N \c@jCol ,
3028     \int_use:N \g_@@_col_total_int
3029   }
3030 }
```

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`).

```
3031 \seq_if_empty:NF \g_@@_pos_of_blocks_seq
3032 {
3033   \tl_gput_right:Nx \g_@@_aux_tl
3034   {
3035     \seq_gset_from_clist:Nn \exp_not:N \g_@@_pos_of_blocks_seq
3036     { \seq_use:Nnnn \g_@@_pos_of_blocks_seq , , , }
3037   }
3038 }
3039 \seq_if_empty:NF \g_@@_multicolumn_cells_seq
3040 {
3041   \tl_gput_right:Nx \g_@@_aux_tl
3042   {
3043     \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_cells_seq
3044     { \seq_use:Nnnn \g_@@_multicolumn_cells_seq , , , }
3045     \seq_gset_from_clist:Nn \exp_not:N \g_@@_multicolumn_sizes_seq
3046     { \seq_use:Nnnn \g_@@_multicolumn_sizes_seq , , , }
3047   }
3048 }
```

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

```
3049 \@@_create_diag_nodes:
```

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

```
3050 \pgfpicture
3051 \int_step_inline:nn \c@iRow
3052 {
3053   \pgfnodealias
3054   { \@@_env: - ##1 - last }
3055   { \@@_env: - ##1 - \int_use:N \c@jCol }
3056 }
3057 \int_step_inline:nn \c@jCol
3058 {
3059   \pgfnodealias
3060   { \@@_env: - last - ##1 }
3061   { \@@_env: - \int_use:N \c@iRow - ##1 }
```

```

3062     }
3063     \str_if_empty:NF \l_@@_name_str
3064     {
3065         \int_step_inline:nn \c@iRow
3066         {
3067             \pgfnodealias
3068             { \l_@@_name_str - ##1 - last }
3069             { \@@_env: - ##1 - \int_use:N \c@jCol }
3070         }
3071         \int_step_inline:nn \c@jCol
3072         {
3073             \pgfnodealias
3074             { \l_@@_name_str - last - ##1 }
3075             { \@@_env: - \int_use:N \c@iRow - ##1 }
3076         }
3077     }
3078     \endpgfpicture

```

By default, the diagonal lines will be parallelized⁷¹. There are two types of diagonals lines: the `\Ddots` diagonals and the `\Iddots` diagonals. We have to count both types in order to know whether a diagonal is the first of its type in the current `{NiceArray}` environment.

```

3079     \bool_if:NT \l_@@_parallelize_diags_bool
3080     {
3081         \int_gzero_new:N \g_@@_ddots_int
3082         \int_gzero_new:N \g_@@_iddots_int

```

The dimensions `\g_@@_delta_x_one_dim` and `\g_@@_delta_y_one_dim` will contain the Δ_x and Δ_y of the first `\Ddots` diagonal. We have to store these values in order to draw the others `\Ddots` diagonals parallel to the first one. Similarly `\g_@@_delta_x_two_dim` and `\g_@@_delta_y_two_dim` are the Δ_x and Δ_y of the first `\Iddots` diagonal.

```

3083         \dim_gzero_new:N \g_@@_delta_x_one_dim
3084         \dim_gzero_new:N \g_@@_delta_y_one_dim
3085         \dim_gzero_new:N \g_@@_delta_x_two_dim
3086         \dim_gzero_new:N \g_@@_delta_y_two_dim
3087     }
3088     \int_zero_new:N \l_@@_initial_i_int
3089     \int_zero_new:N \l_@@_initial_j_int
3090     \int_zero_new:N \l_@@_final_i_int
3091     \int_zero_new:N \l_@@_final_j_int
3092     \bool_set_false:N \l_@@_initial_open_bool
3093     \bool_set_false:N \l_@@_final_open_bool

```

If the option `small` is used, the values `\l_@@_radius_dim` and `\l_@@_inter_dots_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.

```

3094     \bool_if:NT \l_@@_small_bool
3095     {
3096         \dim_set:Nn \l_@@_radius_dim { 0.37 pt }
3097         \dim_set:Nn \l_@@_inter_dots_dim { 0.25 em }

```

The dimension `\l_@@_xdots_shorten_dim` corresponds to the option `xdots/shorten` available to the user. That's why we give a new value according to the current value, and not an absolute value.

```

3098         \dim_set:Nn \l_@@_xdots_shorten_dim { 0.6 \l_@@_xdots_shorten_dim }
3099     }

```

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

```

3100     \@@_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.

⁷¹It's possible to use the option `parallelize-diags` to disable this parallelization.

```
3101 \@@_compute_corners:
```

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

```
3102 \@@_adjust_pos_of_blocks_seq:
3103 \tl_if_empty:NF \l_@@_hlines_clist \@@_draw_hlines:
3104 \tl_if_empty:NF \l_@@_vlines_clist \@@_draw_vlines:
```

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

```
3105 \bool_if:NT \c_@@_tikz_loaded_bool
3106 {
3107   \tikzset
3108   {
3109     every-picture / .style =
3110     {
3111       overlay ,
3112       remember-picture ,
3113       name-prefix = \@@_env: -
3114     }
3115   }
3116 }
3117 \cs_set_eq:NN \ialign \@@_old_ialign:
3118 \cs_set_eq:NN \SubMatrix \@@_SubMatrix
3119 \cs_set_eq:NN \UnderBrace \@@_UnderBrace
3120 \cs_set_eq:NN \OverBrace \@@_OverBrace
3121 \cs_set_eq:NN \line \@@_line
3122 \g_@@_internal_code_after_tl
3123 \tl_gclear:N \g_@@_internal_code_after_tl
```

When `light-syntax` is used, we insert systematically a `\CodeAfter` in the flow. Thus, it’s possible to have two instructions `\CodeAfter` and the second may be in `\g_nicematrix_code_after_tl`. That’s why we set `\Code-after` to be *no-op* now.

```
3124 \cs_set_eq:NN \CodeAfter \prg_do_nothing:
```

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

```
3125 \seq_gclear:N \g_@@_submatrix_names_seq
```

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

```
3126 \exp_last_unbraced:NV \@@_CodeAfter_keys: \g_nicematrix_code_after_tl
3127 \scan_stop:
3128 \tl_gclear:N \g_nicematrix_code_after_tl
3129 \group_end:
```

`\g_nicematrix_code_before_tl` is for instructions in the cells of the array such as `\rowcolor` and `\cellcolor` (when the key `colortbl-like` is in force). These instructions will be written on the `aux` file to be added to the `code-before` in the next run.

```
3130 \tl_if_empty:NF \g_nicematrix_code_before_tl
3131 {
```

The command `\rowcolor` in `tabular` will in fact use `\rectanglecolor` in order to follow the behaviour of `\rowcolor` of `colortbl`. That’s why there may be a command `\rectanglecolor` in `\g_nicematrix_code_before_tl`. In order to avoid an error during the expansion, we define a protected version of `\rectanglecolor`.

```
3132 \cs_set_protected:Npn \rectanglecolor { }
3133 \cs_set_protected:Npn \columncolor { }
3134 \tl_gput_right:Nx \g_@@_aux_tl
3135 {
3136   \tl_gset:Nn \exp_not:N \g_@@_code_before_tl
3137   { \exp_not:V \g_nicematrix_code_before_tl }
3138 }
3139 \bool_set_true:N \l_@@_code_before_bool
```

```

3140     }

3141     \str_gclear:N \g_@@_name_env_str
3142     \@@_restore_iRow_jCol:

```

The command `\CT@arc@` contains the instruction of color for the rules of the array⁷². This command is used by `\CT@arc@` but we use it also for compatibility with `colortbl`. But we want also to be able to use color for the rules of the array when `colortbl` is *not* loaded. That’s why we do the following instruction which is in the patch of the end of arrays done by `colortbl`.

```

3143     \cs_gset_eq:NN \CT@arc@ \@@_old_CT@arc@
3144 }

```

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`.

```

3145 \NewDocumentCommand \@@_CodeAfter_keys: { 0 { } }
3146 { \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).

```

3147 \cs_new_protected:Npn \@@_adjust_pos_of_blocks_seq:
3148 {
3149     \seq_gset_map_x:NNn \g_@@_pos_of_blocks_seq \g_@@_pos_of_blocks_seq
3150     { \@@_adjust_pos_of_blocks_seq_i:nnnnn #1 }
3151 }

```

The following command must *not* be protected.

```

3152 \cs_new:Npn \@@_adjust_pos_of_blocks_seq_i:nnnnn #1 #2 #3 #4 #5
3153 {
3154     { #1 }
3155     { #2 }
3156     {
3157         \int_compare:nNnTF { #3 } > { 99 }
3158         { \int_use:N \c@iRow }
3159         { #3 }
3160     }
3161     {
3162         \int_compare:nNnTF { #4 } > { 99 }
3163         { \int_use:N \c@jCol }
3164         { #4 }
3165     }
3166     { #5 }
3167 }

```

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).

```

3168 \hook_gput_code:nnn { begindocument } { . }
3169 {
3170     \cs_new_protected:Npx \@@_draw_dotted_lines:
3171     {
3172         \c_@@_pgfortikzpicture_tl
3173         \@@_draw_dotted_lines_i:
3174         \c_@@_endpgfortikzpicture_tl

```

⁷²e.g. `\color[rgb]{0.5,0.5,0}`

```

3175     }
3176 }

```

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

```

3177 \cs_new_protected:Npn \@@_draw_dotted_lines_i:
3178 {
3179     \pgfrememberpicturepositiononpagetrue
3180     \pgf@relevantforpicturesizefalse
3181     \g_@@_HVdotsfor_lines_tl
3182     \g_@@_Vdots_lines_tl
3183     \g_@@_Ddots_lines_tl
3184     \g_@@_Iddots_lines_tl
3185     \g_@@_Cdots_lines_tl
3186     \g_@@_Ldots_lines_tl
3187 }

3188 \cs_new_protected:Npn \@@_restore_iRow_jCol:
3189 {
3190     \cs_if_exist:NT \theiRow { \int_gset_eq:NN \c@iRow \l_@@_old_iRow_int }
3191     \cs_if_exist:NT \thejCol { \int_gset_eq:NN \c@jCol \l_@@_old_jCol_int }
3192 }

```

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

```

3193 \pgfdeclareshape { @@_diag_node }
3194 {
3195     \savedanchor { \five }
3196     {
3197         \dim_gset_eq:NN \pgf@x \l_tmpa_dim
3198         \dim_gset_eq:NN \pgf@y \l_tmpb_dim
3199     }
3200     \anchor { 5 } { \five }
3201     \anchor { center } { \pgfpointorigin }
3202 }

```

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).

```

3203 \cs_new_protected:Npn \@@_create_diag_nodes:
3204 {
3205     \pgfpicture
3206     \pgfrememberpicturepositiononpagetrue
3207     \int_step_inline:nn { \int_max:nn \c@iRow \c@jCol }
3208     {
3209         \@@_qpoint:n { col - \int_min:nn { ##1 } { \c@jCol + 1 } }
3210         \dim_set_eq:NN \l_tmpa_dim \pgf@x
3211         \@@_qpoint:n { row - \int_min:nn { ##1 } { \c@iRow + 1 } }
3212         \dim_set_eq:NN \l_tmpb_dim \pgf@y
3213         \@@_qpoint:n { col - \int_min:nn { ##1 + 1 } { \c@jCol + 1 } }
3214         \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
3215         \@@_qpoint:n { row - \int_min:nn { ##1 + 1 } { \c@iRow + 1 } }
3216         \dim_set_eq:NN \l_@@_tmpd_dim \pgf@y
3217         \pgftransformshift { \pgfpoint \l_tmpa_dim \l_tmpb_dim }

```

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

```

3218         \dim_set:Nn \l_tmpa_dim { ( \l_@@_tmpc_dim - \l_tmpa_dim ) / 2 }
3219         \dim_set:Nn \l_tmpb_dim { ( \l_@@_tmpd_dim - \l_tmpb_dim ) / 2 }
3220         \pgfnode { @@_diag_node } { center } { } { \@@_env: - ##1 } { }
3221         \str_if_empty:NF \l_@@_name_str
3222         { \pgfnodealias { \l_@@_name_str - ##1 } { \@@_env: - ##1 } }
3223     }

```



```

3224 \int_set:Nn \l_tmpa_int { \int_max:nn \c@iRow \c@jCol + 1 }
3225 @@_qpoint:n { row - \int_min:nn { \l_tmpa_int } { \c@iRow + 1 } }
3226 \dim_set_eq:NN \l_tmpa_dim \pgf@y
3227 @@_qpoint:n { col - \int_min:nn { \l_tmpa_int } { \c@jCol + 1 } }
3228 \pgfcoordinate
3229 { @@_env: - \int_use:N \l_tmpa_int } { \pgfpoint \pgf@x \l_tmpa_dim }
3230 \pgfnodealias
3231 { @@_env: - last }
3232 { @@_env: - \int_eval:n { \int_max:nn \c@iRow \c@jCol + 1 } }
3233 \str_if_empty:NF \l_@@_name_str
3234 {
3235     \pgfnodealias
3236     { \l_@@_name_str - \int_use:N \l_tmpa_int }
3237     { @@_env: - \int_use:N \l_tmpa_int }
3238     \pgfnodealias
3239     { \l_@@_name_str - last }
3240     { @@_env: - last }
3241 }
3242 \endpgfpicture
3243 }

```

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.

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

- 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.

- `\l_@@_initial_i_int` and `\l_@@_initial_j_int` which are the coordinates of one extremity of the line;
- `\l_@@_final_i_int` and `\l_@@_final_j_int` which are the coordinates of the other extremity of the line;
- `\l_@@_initial_open_bool` and `\l_@@_final_open_bool` to indicate whether the extremities are open or not.

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

Initialization of variables.

```

3247 \int_set:Nn \l_@@_initial_i_int { #1 }
3248 \int_set:Nn \l_@@_initial_j_int { #2 }
3249 \int_set:Nn \l_@@_final_i_int { #1 }
3250 \int_set:Nn \l_@@_final_j_int { #2 }

```

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

```

3251     \bool_set_false:N \l_@@_stop_loop_bool
3252     \bool_do_until:Nn \l_@@_stop_loop_bool
3253     {
3254         \int_add:Nn \l_@@_final_i_int { #3 }
3255         \int_add:Nn \l_@@_final_j_int { #4 }

```

We test if we are still in the matrix.

```

3256     \bool_set_false:N \l_@@_final_open_bool
3257     \int_compare:nNnTF \l_@@_final_i_int > \l_@@_row_max_int
3258     {
3259         \int_compare:nNnTF { #3 } = 1
3260         { \bool_set_true:N \l_@@_final_open_bool }
3261         {
3262             \int_compare:nNnT \l_@@_final_j_int > \l_@@_col_max_int
3263             { \bool_set_true:N \l_@@_final_open_bool }
3264         }
3265     }
3266     {
3267         \int_compare:nNnTF \l_@@_final_j_int < \l_@@_col_min_int
3268         {
3269             \int_compare:nNnT { #4 } = { -1 }
3270             { \bool_set_true:N \l_@@_final_open_bool }
3271         }
3272         {
3273             \int_compare:nNnT \l_@@_final_j_int > \l_@@_col_max_int
3274             {
3275                 \int_compare:nNnT { #4 } = 1
3276                 { \bool_set_true:N \l_@@_final_open_bool }
3277             }
3278         }
3279     }
3280     \bool_if:NTF \l_@@_final_open_bool

```

If we are outside the matrix, we have found the extremity of the dotted line and it’s an *open* extremity.

```

3281     {

```

We do a step backwards.

```

3282         \int_sub:Nn \l_@@_final_i_int { #3 }
3283         \int_sub:Nn \l_@@_final_j_int { #4 }
3284         \bool_set_true:N \l_@@_stop_loop_bool
3285     }

```

If we are in the matrix, we test whether the cell is empty. If it’s not the case, we stop the loop because we have found the correct values for `\l_@@_final_i_int` and `\l_@@_final_j_int`.

```

3286     {
3287         \cs_if_exist:cTF
3288         {
3289             @@ _ dotted _
3290             \int_use:N \l_@@_final_i_int -
3291             \int_use:N \l_@@_final_j_int
3292         }
3293         {
3294             \int_sub:Nn \l_@@_final_i_int { #3 }
3295             \int_sub:Nn \l_@@_final_j_int { #4 }
3296             \bool_set_true:N \l_@@_final_open_bool
3297             \bool_set_true:N \l_@@_stop_loop_bool
3298         }
3299     }
3300     \cs_if_exist:cTF
3301     {
3302         pgf @ sh @ ns @ \@@_env:
3303         - \int_use:N \l_@@_final_i_int

```

```

3304         - \int_use:N \l_@@_final_j_int
3305     }
3306     { \bool_set_true:N \l_@@_stop_loop_bool }

```

If the case is empty, we declare that the cell as non-empty. Indeed, we will draw a dotted line and the cell will be on that dotted line. All the cells of a dotted line have to be marked as “dotted” because we don’t want intersections between dotted lines. We recall that the research of the extremities of the lines are all done in the same TeX group (the group of the environment), even though, when the extremities are found, each line is drawn in a TeX group that we will open for the options of the line.

```

3307     {
3308         \cs_set:cpn
3309         {
3310             @@ _ dotted _
3311             \int_use:N \l_@@_final_i_int -
3312             \int_use:N \l_@@_final_j_int
3313         }
3314         { }
3315     }
3316 }
3317 }
3318 }

```

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

```

3319 \bool_set_false:N \l_@@_stop_loop_bool
3320 \bool_do_until:Nn \l_@@_stop_loop_bool
3321 {
3322     \int_sub:Nn \l_@@_initial_i_int { #3 }
3323     \int_sub:Nn \l_@@_initial_j_int { #4 }
3324     \bool_set_false:N \l_@@_initial_open_bool
3325     \int_compare:nNnTF \l_@@_initial_i_int < \l_@@_row_min_int
3326     {
3327         \int_compare:nNnTF { #3 } = 1
3328         { \bool_set_true:N \l_@@_initial_open_bool }
3329         {
3330             \int_compare:nNnT \l_@@_initial_j_int = { \l_@@_col_min_int -1 }
3331             { \bool_set_true:N \l_@@_initial_open_bool }
3332         }
3333     }
3334     {
3335         \int_compare:nNnTF \l_@@_initial_j_int < \l_@@_col_min_int
3336         {
3337             \int_compare:nNnT { #4 } = 1
3338             { \bool_set_true:N \l_@@_initial_open_bool }
3339         }
3340         {
3341             \int_compare:nNnT \l_@@_initial_j_int > \l_@@_col_max_int
3342             {
3343                 \int_compare:nNnT { #4 } = { -1 }
3344                 { \bool_set_true:N \l_@@_initial_open_bool }
3345             }
3346         }
3347     }
3348     \bool_if:NTF \l_@@_initial_open_bool
3349     {
3350         \int_add:Nn \l_@@_initial_i_int { #3 }
3351         \int_add:Nn \l_@@_initial_j_int { #4 }
3352         \bool_set_true:N \l_@@_stop_loop_bool
3353     }
3354     {
3355         \cs_if_exist:cTF
3356         {

```

```

3357         @@ _ dotted _
3358         \int_use:N \l_@@_initial_i_int -
3359         \int_use:N \l_@@_initial_j_int
3360     }
3361     {
3362         \int_add:Nn \l_@@_initial_i_int { #3 }
3363         \int_add:Nn \l_@@_initial_j_int { #4 }
3364         \bool_set_true:N \l_@@_initial_open_bool
3365         \bool_set_true:N \l_@@_stop_loop_bool
3366     }
3367     {
3368         \cs_if_exist:cTF
3369         {
3370             pgf @ sh @ ns @ \@@_env:
3371             - \int_use:N \l_@@_initial_i_int
3372             - \int_use:N \l_@@_initial_j_int
3373         }
3374         { \bool_set_true:N \l_@@_stop_loop_bool }
3375         {
3376             \cs_set:cpn
3377             {
3378                 @@ _ dotted _
3379                 \int_use:N \l_@@_initial_i_int -
3380                 \int_use:N \l_@@_initial_j_int
3381             }
3382             { }
3383         }
3384     }
3385 }
3386 }

```

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.

```

3387     \seq_gput_right:Nx \g_@@_pos_of_xdots_seq
3388     {
3389         { \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`.

```

3390         { \int_min:nn \l_@@_initial_j_int \l_@@_final_j_int }
3391         { \int_use:N \l_@@_final_i_int }
3392         { \int_max:nn \l_@@_initial_j_int \l_@@_final_j_int }
3393         { } % for the name of the block
3394     }
3395 }

```

The following command (*when it will be written*) will set the four counters `\l_@@_row_min_int`, `\l_@@_row_max_int`, `\l_@@_col_min_int` and `\l_@@_col_max_int` to the intersections of the submatrices which contains the cell of row #1 and column #2. As of now, it’s only the whole array (excepted exterior rows and columns).

```

3396 \cs_new_protected:Npn \@@_adjust_to_submatrix:nn #1 #2
3397 {
3398     \int_set:Nn \l_@@_row_min_int 1
3399     \int_set:Nn \l_@@_col_min_int 1
3400     \int_set_eq:NN \l_@@_row_max_int \c@iRow
3401     \int_set_eq:NN \l_@@_col_max_int \c@jCol

```

We do a loop over all the submatrices specified in the code-before. We have stored the position of all those submatrices in `\g_@@_submatrix_seq`.

```

3402     \seq_map_inline:Nn \g_@@_submatrix_seq
3403     { \@@_adjust_to_submatrix:nnnnn { #1 } { #2 } ##1 }
3404 }

```

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

```

3405 \cs_set_protected:Npn \@@_adjust_to_submatrix:nnnnnn #1 #2 #3 #4 #5 #6
3406 {
3407   \bool_if:nT
3408   {
3409     \int_compare_p:n { #3 <= #1 }
3410     && \int_compare_p:n { #1 <= #5 }
3411     && \int_compare_p:n { #4 <= #2 }
3412     && \int_compare_p:n { #2 <= #6 }
3413   }
3414   {
3415     \int_set:Nn \l_@@_row_min_int { \int_max:nn \l_@@_row_min_int { #3 } }
3416     \int_set:Nn \l_@@_col_min_int { \int_max:nn \l_@@_col_min_int { #4 } }
3417     \int_set:Nn \l_@@_row_max_int { \int_min:nn \l_@@_row_max_int { #5 } }
3418     \int_set:Nn \l_@@_col_max_int { \int_min:nn \l_@@_col_max_int { #6 } }
3419   }
3420 }

3421 \cs_new_protected:Npn \@@_set_initial_coords:
3422 {
3423   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
3424   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
3425 }
3426 \cs_new_protected:Npn \@@_set_final_coords:
3427 {
3428   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
3429   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
3430 }
3431 \cs_new_protected:Npn \@@_set_initial_coords_from_anchor:n #1
3432 {
3433   \pgfpointanchor
3434   {
3435     \@@_env:
3436     - \int_use:N \l_@@_initial_i_int
3437     - \int_use:N \l_@@_initial_j_int
3438   }
3439   { #1 }
3440   \@@_set_initial_coords:
3441 }
3442 \cs_new_protected:Npn \@@_set_final_coords_from_anchor:n #1
3443 {
3444   \pgfpointanchor
3445   {
3446     \@@_env:
3447     - \int_use:N \l_@@_final_i_int
3448     - \int_use:N \l_@@_final_j_int
3449   }
3450   { #1 }
3451   \@@_set_final_coords:
3452 }

3453 \cs_new_protected:Npn \@@_open_x_initial_dim:
3454 {
3455   \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
3456   \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
3457   {
3458     \cs_if_exist:cT
3459     { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
3460     {
3461       \pgfpointanchor
3462       { \@@_env: - ##1 - \int_use:N \l_@@_initial_j_int }
3463       { west }

```

```

3464         \dim_set:Nn \l_@@_x_initial_dim
3465         { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
3466     }
3467 }

```

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

```

3468     \dim_compare:nNnT \l_@@_x_initial_dim = \c_max_dim
3469     {
3470         \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
3471         \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
3472         \dim_add:Nn \l_@@_x_initial_dim \col@sep
3473     }
3474 }

3475 \cs_new_protected:Npn \@@_open_x_final_dim:
3476 {
3477     \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
3478     \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
3479     {
3480         \cs_if_exist:cT
3481         { \pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
3482         {
3483             \pgfpointanchor
3484             { \@@_env: - ##1 - \int_use:N \l_@@_final_j_int }
3485             { east }
3486             \dim_set:Nn \l_@@_x_final_dim
3487             { \dim_max:nn \l_@@_x_final_dim \pgf@x }
3488         }
3489     }

```

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

```

3490     \dim_compare:nNnT \l_@@_x_final_dim = { - \c_max_dim }
3491     {
3492         \@@_qpoint:n { col - \@@_succ:n \l_@@_final_j_int }
3493         \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
3494         \dim_sub:Nn \l_@@_x_final_dim \col@sep
3495     }
3496 }

```

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.

```

3497 \cs_new_protected:Npn \@@_draw_Ldots:nnn #1 #2 #3
3498 {
3499     \@@_adjust_to_submatrix:nn { #1 } { #2 }
3500     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
3501     {
3502         \@@_find_extremities_of_line:nnnn { #1 } { #2 } 0 1

```

The previous command may have changed the current environment by marking some cells as “dotted”, but, fortunately, it is outside the group for the options of the line.

```

3503     \group_begin:
3504     \int_compare:nNnTF { #1 } = 0
3505     { \color { nicematrix-first-row } }
3506     {

```

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.

```

3507         \int_compare:nNnT { #1 } = \l_@@_last_row_int
3508         { \color { nicematrix-last-row } }
3509     }
3510     \keys_set:nn { NiceMatrix / xdots } { #3 }
3511     \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
3512     \@@_actually_draw_Ldots:
3513 \group_end:

```

```

3514     }
3515 }

```

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`.

```

3516 \cs_new_protected:Npn \@@_actually_draw_Ldots:
3517 {
3518   \bool_if:NTF \l_@@_initial_open_bool
3519   {
3520     \@@_open_x_initial_dim:
3521     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
3522     \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
3523   }
3524   { \@@_set_initial_coords_from_anchor:n { base-east } }
3525   \bool_if:NTF \l_@@_final_open_bool
3526   {
3527     \@@_open_x_final_dim:
3528     \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
3529     \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
3530   }
3531   { \@@_set_final_coords_from_anchor:n { base-west } }

```

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

```

3532   \dim_add:Nn \l_@@_y_initial_dim \l_@@_radius_dim
3533   \dim_add:Nn \l_@@_y_final_dim \l_@@_radius_dim
3534   \@@_draw_line:
3535 }

```

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.

```

3536 \cs_new_protected:Npn \@@_draw_Cdots:nnn #1 #2 #3
3537 {
3538   \@@_adjust_to_submatrix:nn { #1 } { #2 }
3539   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
3540   {
3541     \@@_find_extremities_of_line:nnnn { #1 } { #2 } 0 1

```

The previous command may have changed the current environment by marking some cells as “dotted”, but, fortunately, it is outside the group for the options of the line.

```

3542   \group_begin:
3543   \int_compare:nNnTF { #1 } = 0
3544   { \color { nicematrix-first-row } }
3545   {

```

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.

```

3546     \int_compare:nNnT { #1 } = \l_@@_last_row_int
3547     { \color { nicematrix-last-row } }
3548   }
3549   \keys_set:nn { NiceMatrix / xdots } { #3 }

```

```

3550         \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
3551         \@@_actually_draw_Cdots:
3552     \group_end:
3553 }
3554 }

```

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.`

```

3555 \cs_new_protected:Npn \@@_actually_draw_Cdots:
3556 {
3557     \bool_if:NTF \l_@@_initial_open_bool
3558     { \@@_open_x_initial_dim: }
3559     { \@@_set_initial_coords_from_anchor:n { mid-east } }
3560     \bool_if:NTF \l_@@_final_open_bool
3561     { \@@_open_x_final_dim: }
3562     { \@@_set_final_coords_from_anchor:n { mid-west } }
3563     \bool_lazy_and:nnTF
3564     \l_@@_initial_open_bool
3565     \l_@@_final_open_bool
3566     {
3567         \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int }
3568         \dim_set_eq:NN \l_tmpa_dim \pgf@y
3569         \@@_qpoint:n { row - \@@_succ:n \l_@@_initial_i_int }
3570         \dim_set:Nn \l_@@_y_initial_dim { ( \l_tmpa_dim + \pgf@y ) / 2 }
3571         \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim
3572     }
3573     {
3574         \bool_if:NT \l_@@_initial_open_bool
3575         { \dim_set_eq:NN \l_@@_y_initial_dim \l_@@_y_final_dim }
3576         \bool_if:NT \l_@@_final_open_bool
3577         { \dim_set_eq:NN \l_@@_y_final_dim \l_@@_y_initial_dim }
3578     }
3579     \@@_draw_line:
3580 }
3581 \cs_new_protected:Npn \@@_open_y_initial_dim:
3582 {
3583     \@@_qpoint:n { row - \int_use:N \l_@@_initial_i_int - base }
3584     \dim_set:Nn \l_@@_y_initial_dim
3585     { \pgf@y + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch }
3586     \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
3587     {
3588         \cs_if_exist:cT
3589         { \pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
3590         {
3591             \pgfpointanchor
3592             { \@@_env: - \int_use:N \l_@@_initial_i_int - ##1 }
3593             { north }
3594             \dim_set:Nn \l_@@_y_initial_dim
3595             { \dim_max:nn \l_@@_y_initial_dim \pgf@y }
3596         }
3597     }
3598 }

```



```

3599 \cs_new_protected:Npn \@@_open_y_final_dim:
3600 {
3601   \@@_qpoint:n { row - \int_use:N \l_@@_final_i_int - base }
3602   \dim_set:Nn \l_@@_y_final_dim
3603   { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch }
3604   \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
3605   {
3606     \cs_if_exist:cT
3607     { \pgf @ sh @ ns @ \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
3608     {
3609       \pgfpointanchor
3610       { \@@_env: - \int_use:N \l_@@_final_i_int - ##1 }
3611       { south }
3612       \dim_set:Nn \l_@@_y_final_dim
3613       { \dim_min:nn \l_@@_y_final_dim \pgf@y }
3614     }
3615   }
3616 }

```

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.

```

3617 \cs_new_protected:Npn \@@_draw_Vdots:nnn #1 #2 #3
3618 {
3619   \@@_adjust_to_submatrix:nn { #1 } { #2 }
3620   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
3621   {
3622     \@@_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.

```

3623   \group_begin:
3624   \int_compare:nNnTF { #2 } = 0
3625   { \color { nicematrix-first-col } }
3626   {
3627     \int_compare:nNnT { #2 } = \l_@@_last_col_int
3628     { \color { nicematrix-last-col } }
3629   }
3630   \keys_set:nn { NiceMatrix / xdots } { #3 }
3631   \tl_if_empty:VF \l_@@_xdots_color_tl
3632   { \color { \l_@@_xdots_color_tl } }
3633   \@@_actually_draw_Vdots:
3634   \group_end:
3635 }
3636 }

```

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`.

```

3637 \cs_new_protected:Npn \@@_actually_draw_Vdots:
3638 {

```

The boolean `\l_tmpa_bool` indicates whether the column is of type 1 or may be considered as if.

```

3639   \bool_set_false:N \l_tmpa_bool

```

First the case when the line is closed on both ends.

```

3640 \bool_lazy_or:nnF \l_@@_initial_open_bool \l_@@_final_open_bool
3641 {
3642   \@@_set_initial_coords_from_anchor:n { south-west }
3643   \@@_set_final_coords_from_anchor:n { north-west }
3644   \bool_set:Nn \l_tmpa_bool
3645     { \dim_compare_p:nNn \l_@@_x_initial_dim = \l_@@_x_final_dim }
3646 }

```

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

```

3647 \bool_if:NTF \l_@@_initial_open_bool
3648   \@@_open_y_initial_dim:
3649   { \@@_set_initial_coords_from_anchor:n { south } }
3650 \bool_if:NTF \l_@@_final_open_bool
3651   \@@_open_y_final_dim:
3652   { \@@_set_final_coords_from_anchor:n { north } }
3653 \bool_if:NTF \l_@@_initial_open_bool
3654 {
3655   \bool_if:NTF \l_@@_final_open_bool
3656   {
3657     \@@_qpoint:n { col - \int_use:N \l_@@_initial_j_int }
3658     \dim_set_eq:NN \l_tmpa_dim \pgf@x
3659     \@@_qpoint:n { col - \@@_succ:n \l_@@_initial_j_int }
3660     \dim_set:Nn \l_@@_x_initial_dim { ( \pgf@x + \l_tmpa_dim ) / 2 }
3661     \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim

```

We may think that the final user won't use a "last column" which contains only a command `\Vdots`. However, if the `\Vdots` is in fact used to draw, not a dotted line, but an arrow (to indicate the number of rows of the matrix), it may be really encountered.

```

3662 \int_compare:nNnT \l_@@_last_col_int > { -2 }
3663 {
3664   \int_compare:nNnT \l_@@_initial_j_int = \g_@@_col_total_int
3665   {
3666     \dim_set_eq:NN \l_tmpa_dim \l_@@_right_margin_dim
3667     \dim_add:Nn \l_tmpa_dim \l_@@_extra_right_margin_dim
3668     \dim_add:Nn \l_@@_x_initial_dim \l_tmpa_dim
3669     \dim_add:Nn \l_@@_x_final_dim \l_tmpa_dim
3670   }
3671 }
3672 }
3673 { \dim_set_eq:NN \l_@@_x_initial_dim \l_@@_x_final_dim }
3674 }
3675 {
3676   \bool_if:NTF \l_@@_final_open_bool
3677   { \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim }
3678 }

```

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

```

3679 \dim_compare:nNnF \l_@@_x_initial_dim = \l_@@_x_final_dim
3680 {
3681   \dim_set:Nn \l_@@_x_initial_dim
3682   {
3683     \bool_if:NTF \l_tmpa_bool \dim_min:nn \dim_max:nn
3684       \l_@@_x_initial_dim \l_@@_x_final_dim
3685   }
3686   \dim_set_eq:NN \l_@@_x_final_dim \l_@@_x_initial_dim
3687 }
3688 }
3689 }
3690 \@@_draw_line:
3691 }

```

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.

```

3692 \cs_new_protected:Npn \@@_draw_Ddots:nnn #1 #2 #3
3693 {
3694   \@@_adjust_to_submatrix:nn { #1 } { #2 }
3695   \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
3696   {
3697     \@@_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.

```

3698     \group_begin:
3699     \keys_set:nn { NiceMatrix / xdots } { #3 }
3700     \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
3701     \@@_actually_draw_Ddots:
3702   \group_end:
3703 }
3704 }

```

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.`

```

3705 \cs_new_protected:Npn \@@_actually_draw_Ddots:
3706 {
3707   \bool_if:NTF \l_@@_initial_open_bool
3708   {
3709     \@@_open_y_initial_dim:
3710     \@@_open_x_initial_dim:
3711   }
3712   { \@@_set_initial_coords_from_anchor:n { south-east } }
3713   \bool_if:NTF \l_@@_final_open_bool
3714   {
3715     \@@_open_x_final_dim:
3716     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
3717   }
3718   { \@@_set_final_coords_from_anchor:n { north-west } }

```

We have retrieved the coordinates in the usual way (they are stored in `\l_@@_x_initial_dim`, etc.). If the parallelization of the diagonals is set, we will have (maybe) to adjust the fourth coordinate.

```

3719   \bool_if:NT \l_@@_parallelize_diags_bool
3720   {
3721     \int_gincr:N \g_@@_ddots_int

```

We test if the diagonal line is the first one (the counter `\g_@@_ddots_int` is created for this usage).

```

3722     \int_compare:nNnTF \g_@@_ddots_int = 1

```

If the diagonal line is the first one, we have no adjustment of the line to do but we store the Δ_x and the Δ_y of the line because these values will be used to draw the others diagonal lines parallels to the first one.

```

3723     {
3724       \dim_gset:Nn \g_@@_delta_x_one_dim

```

```

3725         { \l_@@_x_final_dim - \l_@@_x_initial_dim }
3726         \dim_gset:Nn \g_@@_delta_y_one_dim
3727         { \l_@@_y_final_dim - \l_@@_y_initial_dim }
3728     }

```

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`.

```

3729     {
3730         \dim_set:Nn \l_@@_y_final_dim
3731         {
3732             \l_@@_y_initial_dim +
3733             ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
3734             \dim_ratio:nn \g_@@_delta_y_one_dim \g_@@_delta_x_one_dim
3735         }
3736     }
3737 }
3738 \@@_draw_line:
3739 }

```

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.

```

3740 \cs_new_protected:Npn \@@_draw_Iddots:nnn #1 #2 #3
3741 {
3742     \@@_adjust_to_submatrix:nn { #1 } { #2 }
3743     \cs_if_free:cT { @@ _ dotted _ #1 - #2 }
3744     {
3745         \@@_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.

```

3746     \group_begin:
3747         \keys_set:nn { NiceMatrix / xdots } { #3 }
3748         \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
3749         \@@_actually_draw_Iddots:
3750     \group_end:
3751 }
3752 }

```

The command `\@@_actually_draw_Iddots:` has the following implicit arguments:

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

```

3753 \cs_new_protected:Npn \@@_actually_draw_Iddots:
3754 {
3755     \bool_if:NTF \l_@@_initial_open_bool
3756     {
3757         \@@_open_y_initial_dim:
3758         \@@_open_x_initial_dim:
3759     }
3760     { \@@_set_initial_coords_from_anchor:n { south-west } }
3761     \bool_if:NTF \l_@@_final_open_bool
3762     {
3763         \@@_open_y_final_dim:
3764         \@@_open_x_final_dim:

```

```

3765     }
3766     { \l_@@_set_final_coords_from_anchor:n { north~east } }
3767     \bool_if:NT \l_@@_parallelize_diags_bool
3768     {
3769         \int_gincr:N \g_@@_iddots_int
3770         \int_compare:nNnTF \g_@@_iddots_int = 1
3771         {
3772             \dim_gset:Nn \g_@@_delta_x_two_dim
3773             { \l_@@_x_final_dim - \l_@@_x_initial_dim }
3774             \dim_gset:Nn \g_@@_delta_y_two_dim
3775             { \l_@@_y_final_dim - \l_@@_y_initial_dim }
3776         }
3777         {
3778             \dim_set:Nn \l_@@_y_final_dim
3779             {
3780                 \l_@@_y_initial_dim +
3781                 ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
3782                 \dim_ratio:nn \g_@@_delta_y_two_dim \g_@@_delta_x_two_dim
3783             }
3784         }
3785     }
3786     \l_@@_draw_line:
3787 }

```

The actual instructions for drawing the dotted lines with Tikz

The command `\l_@@_draw_line:` should be used in a `{pgfpicture}`. It has six implicit arguments:

- `\l_@@_x_initial_dim`
- `\l_@@_y_initial_dim`
- `\l_@@_x_final_dim`
- `\l_@@_y_final_dim`
- `\l_@@_initial_open_bool`
- `\l_@@_final_open_bool`

```

3788 \cs_new_protected:Npn \l_@@_draw_line:
3789 {
3790     \pgfrememberpicturepositiononpagetrue
3791     \pgf@relevantforpicturesizefalse
3792     \bool_lazy_or:nnTF
3793     { \tl_if_eq_p:NN \l_@@_xdots_line_style_tl \c_@@_standard_tl }

```

The boolean `\l_@@_dotted_bool` is raised for the rules specified by either `\hdottedline` or `:` (or the letter specified by `letter-for-dotted-lines`) in the preamble of the array.

```

3794     \l_@@_dotted_bool
3795     \l_@@_draw_standard_dotted_line:
3796     \l_@@_draw_unstandard_dotted_line:
3797 }

```

We have to do a special construction with `\exp_args:NV` to be able to put in the list of options in the correct place in the Tikz instruction.

```

3798 \cs_new_protected:Npn \l_@@_draw_unstandard_dotted_line:
3799 {
3800     \begin { scope }
3801     \exp_args:No \l_@@_draw_unstandard_dotted_line:n
3802     { \l_@@_xdots_line_style_tl , \l_@@_xdots_color_tl }
3803 }

```

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.

```

3804 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:n #1
3805 {
3806   \@@_draw_unstandard_dotted_line:nVV
3807   { #1 }
3808   \l_@@_xdots_up_tl
3809   \l_@@_xdots_down_tl
3810 }

3811 \cs_new_protected:Npn \@@_draw_unstandard_dotted_line:nnn #1 #2 #3
3812 {
3813   \draw
3814   [
3815     #1 ,
3816     shorten~> = \l_@@_xdots_shorten_dim ,
3817     shorten~< = \l_@@_xdots_shorten_dim ,
3818   ]
3819   ( \l_@@_x_initial_dim , \l_@@_y_initial_dim )

```

Be careful: We can't put `\c_math_toggle_token` instead of `$` in the following lines because we are in the contents of Tikz nodes (and they will be *rescanned* if the Tikz library `babel` is loaded).

```

3820   -- node [ sloped , above ] { $ \scriptstyle #2 $ }
3821   node [ sloped , below ] { $ \scriptstyle #3 $ }
3822   ( \l_@@_x_final_dim , \l_@@_y_final_dim ) ;
3823   \end { scope }
3824 }
3825 \cs_generate_variant:Nn \@@_draw_unstandard_dotted_line:nnn { n V V }

```

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

```

3826 \cs_new_protected:Npn \@@_draw_standard_dotted_line:
3827 {
3828   \bool_lazy_and:nnF
3829   { \tl_if_empty_p:N \l_@@_xdots_up_tl }
3830   { \tl_if_empty_p:N \l_@@_xdots_down_tl }
3831   {
3832     \pgfscope
3833     \pgftransformshift
3834     {
3835       \pgfpointlineattime { 0.5 }
3836       { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
3837       { \pgfpoint \l_@@_x_final_dim \l_@@_y_final_dim }
3838     }
3839     \pgftransformrotate
3840     {
3841       \fp_eval:n
3842       {
3843         atand
3844         (
3845           \l_@@_y_final_dim - \l_@@_y_initial_dim ,
3846           \l_@@_x_final_dim - \l_@@_x_initial_dim
3847         )
3848       }
3849     }
3850     \pgfnode
3851     { rectangle }
3852     { south }
3853     {
3854       \c_math_toggle_token
3855       \scriptstyle \l_@@_xdots_up_tl
3856       \c_math_toggle_token

```

```

3857     }
3858     { }
3859     { \pgfusepath { } }
3860     \pgfnode
3861     { rectangle }
3862     { north }
3863     {
3864         \c_math_toggle_token
3865         \scriptstyle \l_@@_xdots_down_tl
3866         \c_math_toggle_token
3867     }
3868     { }
3869     { \pgfusepath { } }
3870     \endpgfscope
3871 }
3872 \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.

```

3873     \dim_zero_new:N \l_@@_l_dim
3874     \dim_set:Nn \l_@@_l_dim
3875     {
3876         \fp_to_dim:n
3877         {
3878             sqrt
3879             (
3880                 ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) ^ 2
3881                 +
3882                 ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) ^ 2
3883             )
3884         }
3885     }

```

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.

```

3886     \bool_lazy_or:nnF
3887     { \dim_compare_p:nNn { \dim_abs:n \l_@@_l_dim } > \c_@@_max_l_dim }
3888     { \dim_compare_p:nNn \l_@@_l_dim = \c_zero_dim }
3889     \@@_draw_standard_dotted_line_i:
3890     \group_end:
3891 }
3892 \dim_const:Nn \c_@@_max_l_dim { 50 cm }
3893 \cs_new_protected:Npn \@@_draw_standard_dotted_line_i:
3894 {

```

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

```

3895     \bool_if:NTF \l_@@_initial_open_bool
3896     {
3897         \bool_if:NTF \l_@@_final_open_bool
3898         {
3899             \int_set:Nn \l_tmpa_int
3900             { \dim_ratio:nn \l_@@_l_dim \l_@@_inter_dots_dim }
3901         }
3902         {
3903             \int_set:Nn \l_tmpa_int
3904             {
3905                 \dim_ratio:nn
3906                 { \l_@@_l_dim - \l_@@_xdots_shorten_dim }
3907                 \l_@@_inter_dots_dim
3908             }
3909         }

```

```

3910 }
3911 {
3912   \bool_if:NTF \l_@@_final_open_bool
3913   {
3914     \int_set:Nn \l_tmpa_int
3915     {
3916       \dim_ratio:nn
3917       { \l_@@_l_dim - \l_@@_xdots_shorten_dim }
3918       \l_@@_inter_dots_dim
3919     }
3920   }
3921   {
3922     \int_set:Nn \l_tmpa_int
3923     {
3924       \dim_ratio:nn
3925       { \l_@@_l_dim - 2 \l_@@_xdots_shorten_dim }
3926       \l_@@_inter_dots_dim
3927     }
3928   }
3929 }

```

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

```

3930 \dim_set:Nn \l_tmpa_dim
3931 {
3932   ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
3933   \dim_ratio:nn \l_@@_inter_dots_dim \l_@@_l_dim
3934 }
3935 \dim_set:Nn \l_tmpb_dim
3936 {
3937   ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
3938   \dim_ratio:nn \l_@@_inter_dots_dim \l_@@_l_dim
3939 }

```

The length ℓ is the length of the dotted line. We note Δ the length between two dots and n the number of intervals between dots. We note $\delta = \frac{1}{2}(\ell - n\Delta)$. The distance between the initial extremity of the line and the first dot will be equal to $k \cdot \delta$ where $k = 0, 1$ or 2 . We first compute this number k in `\l_tmpb_int`.

```

3940 \int_set:Nn \l_tmpb_int
3941 {
3942   \bool_if:NTF \l_@@_initial_open_bool
3943   { \bool_if:NTF \l_@@_final_open_bool 1 0 }
3944   { \bool_if:NTF \l_@@_final_open_bool 2 1 }
3945 }

```

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

```

3946 \dim_gadd:Nn \l_@@_x_initial_dim
3947 {
3948   ( \l_@@_x_final_dim - \l_@@_x_initial_dim ) *
3949   \dim_ratio:nn
3950   { \l_@@_l_dim - \l_@@_inter_dots_dim * \l_tmpa_int }
3951   { 2 \l_@@_l_dim }
3952   * \l_tmpb_int
3953 }
3954 \dim_gadd:Nn \l_@@_y_initial_dim
3955 {
3956   ( \l_@@_y_final_dim - \l_@@_y_initial_dim ) *
3957   \dim_ratio:nn
3958   { \l_@@_l_dim - \l_@@_inter_dots_dim * \l_tmpa_int }
3959   { 2 \l_@@_l_dim }
3960   * \l_tmpb_int
3961 }
3962 \pgf@relevantforpicturesizefalse

```



```

3963 \int_step_inline:nnn 0 \l_tmpa_int
3964 {
3965   \pgfpathcircle
3966   { \pgfpoint \l_@@_x_initial_dim \l_@@_y_initial_dim }
3967   { \l_@@_radius_dim }
3968   \dim_add:Nn \l_@@_x_initial_dim \l_tmpa_dim
3969   \dim_add:Nn \l_@@_y_initial_dim \l_tmpb_dim
3970 }
3971 \pgfusepathqfill
3972 }

```

User commands available in the new environments

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

The syntax of these commands uses the character `_` as embellishment and that's why we have to insert a character `_` in the *arg spec* of these commands. However, we don't know the future catcode of `_` in the main document (maybe the user will use `underscore`, and, in that case, the catcode is 13 because `underscore` activates `_`). That's why these commands will be defined in a `\hook_gput_code:nnn { begindocument } { . }` and the *arg spec* will be rescanned.

```

3973 \hook_gput_code:nnn { begindocument } { . }
3974 {
3975   \tl_set:Nn \l_@@_argspec_tl { 0 { } E { _ ^ } { { } { } } }
3976   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
3977   \exp_args:NNV \NewDocumentCommand \@@_Ldots \l_@@_argspec_tl
3978   {
3979     \int_compare:nNnTF \c@jCol = 0
3980     { \@@_error:nn { in~first~col } \Ldots }
3981     {
3982       \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
3983       { \@@_error:nn { in~last~col } \Ldots }
3984       {
3985         \@@_instruction_of_type:nnn \c_false_bool { Ldots }
3986         { #1 , down = #2 , up = #3 }
3987       }
3988     }
3989     \bool_if:NF \l_@@_nullify_dots_bool
3990     { \phantom { \ensuremath { \@@_old_ldots } } }
3991     \bool_gset_true:N \g_@@_empty_cell_bool
3992   }

3993   \exp_args:NNV \NewDocumentCommand \@@_Cdots \l_@@_argspec_tl
3994   {
3995     \int_compare:nNnTF \c@jCol = 0
3996     { \@@_error:nn { in~first~col } \Cdots }
3997     {
3998       \int_compare:nNnTF \c@jCol = \l_@@_last_col_int
3999       { \@@_error:nn { in~last~col } \Cdots }
4000       {
4001         \@@_instruction_of_type:nnn \c_false_bool { Cdots }
4002         { #1 , down = #2 , up = #3 }
4003       }
4004     }
4005     \bool_if:NF \l_@@_nullify_dots_bool
4006     { \phantom { \ensuremath { \@@_old_cdots } } }
4007     \bool_gset_true:N \g_@@_empty_cell_bool
4008   }

```

```

4009 \exp_args:NNV \NewDocumentCommand \@@_Vdots \l_@@_argspec_tl
4010 {
4011   \int_compare:nNnTF \c@iRow = 0
4012   { \@@_error:nn { in~first~row } \Vdots }
4013   {
4014     \int_compare:nNnTF \c@iRow = \l_@@_last_row_int
4015     { \@@_error:nn { in~last~row } \Vdots }
4016     {
4017       \@@_instruction_of_type:nnn \c_false_bool { \Vdots }
4018       { #1 , down = #2 , up = #3 }
4019     }
4020   }
4021   \bool_if:NF \l_@@_nullify_dots_bool
4022   { \phantom { \ensuremath { \@@_old_vdots } } }
4023   \bool_gset_true:N \g_@@_empty_cell_bool
4024 }

4025 \exp_args:NNV \NewDocumentCommand \@@_Ddots \l_@@_argspec_tl
4026 {
4027   \int_case:nnF \c@iRow
4028   {
4029     0 { \@@_error:nn { in~first~row } \Ddots }
4030     \l_@@_last_row_int { \@@_error:nn { in~last~row } \Ddots }
4031   }
4032   {
4033     \int_case:nnF \c@jCol
4034     {
4035       0 { \@@_error:nn { in~first~col } \Ddots }
4036       \l_@@_last_col_int { \@@_error:nn { in~last~col } \Ddots }
4037     }
4038     {
4039       \keys_set_known:nn { NiceMatrix / Ddots } { #1 }
4040       \@@_instruction_of_type:nnn \l_@@_draw_first_bool { \Ddots }
4041       { #1 , down = #2 , up = #3 }
4042     }
4043   }
4044 }
4045 \bool_if:NF \l_@@_nullify_dots_bool
4046 { \phantom { \ensuremath { \@@_old_ddots } } }
4047 \bool_gset_true:N \g_@@_empty_cell_bool
4048 }

4049 \exp_args:NNV \NewDocumentCommand \@@_Iddots \l_@@_argspec_tl
4050 {
4051   \int_case:nnF \c@iRow
4052   {
4053     0 { \@@_error:nn { in~first~row } \Iddots }
4054     \l_@@_last_row_int { \@@_error:nn { in~last~row } \Iddots }
4055   }
4056   {
4057     \int_case:nnF \c@jCol
4058     {
4059       0 { \@@_error:nn { in~first~col } \Iddots }
4060       \l_@@_last_col_int { \@@_error:nn { in~last~col } \Iddots }
4061     }
4062     {
4063       \keys_set_known:nn { NiceMatrix / Ddots } { #1 }
4064       \@@_instruction_of_type:nnn \l_@@_draw_first_bool { \Iddots }
4065       { #1 , down = #2 , up = #3 }
4066     }
4067   }
4068   \bool_if:NF \l_@@_nullify_dots_bool

```

```

4069         { \phantom { \ensuremath { \@@_old_iddots } } }
4070         \bool_gset_true:N \g_@@_empty_cell_bool
4071     }
4072 }

```

End of the \AddToHook.

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

```

4073 \keys_define:nn { NiceMatrix / Ddots }
4074 {
4075     draw-first .bool_set:N = \l_@@_draw_first_bool ,
4076     draw-first .default:n = true ,
4077     draw-first .value_forbidden:n = true
4078 }

```

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

```

4079 \cs_new_protected:Npn \@@_Hspace:
4080 {
4081     \bool_gset_true:N \g_@@_empty_cell_bool
4082     \hspace
4083 }

```

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.

```

4084 \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.

```

4085 \cs_new:Npn \@@_Hdotsfor:
4086 {
4087     \bool_lazy_and:nnTF
4088     { \int_compare_p:nNn \c@jCol = 0 }
4089     { \int_compare_p:nNn \l_@@_first_col_int = 0 }
4090     {
4091         \bool_if:NTF \g_@@_after_col_zero_bool
4092         {
4093             \multicolumn { 1 } { c } { }
4094             \@@_Hdotsfor_i
4095         }
4096         { \@@_fatal:n { Hdotsfor~in~col~0 } }
4097     }
4098     {
4099         \multicolumn { 1 } { c } { }
4100         \@@_Hdotsfor_i
4101     }
4102 }

```

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:).

```

4103 \hook_gput_code:nnn { begindocument } { . }
4104 {
4105     \tl_set:Nn \l_@@_argspec_tl { 0 { } m 0 { } E { _ ^ } { { } { } } }
4106     \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl

```

We don't put ! before the last optionnal argument for homogeneity with \Cdots, etc. which have only one optional argument.

```

4107     \exp_args:NNV \NewDocumentCommand \@@_Hdotsfor_i \l_@@_argspec_tl
4108     {
4109         \tl_gput_right:Nx \g_@@_HVdotsfor_lines_tl

```

```

4110     {
4111         \@@_Hdotsfor:nnnn
4112         { \int_use:N \c@iRow }
4113         { \int_use:N \c@jCol }
4114         { #2 }
4115         {
4116             #1 , #3 ,
4117             down = \exp_not:n { #4 } ,
4118             up = \exp_not:n { #5 }
4119         }
4120     }
4121     \prg_replicate:nn { #2 - 1 } { & \multicolumn { 1 } { c } { } }
4122 }
4123 }

```

Enf of \AddToHook.

```

4124 \cs_new_protected:Npn \@@_Hdotsfor:nnnn #1 #2 #3 #4
4125 {
4126     \bool_set_false:N \l_@@_initial_open_bool
4127     \bool_set_false:N \l_@@_final_open_bool

```

For the row, it's easy.

```

4128     \int_set:Nn \l_@@_initial_i_int { #1 }
4129     \int_set_eq:NN \l_@@_final_i_int \l_@@_initial_i_int

```

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

```

4130     \int_compare:nNnTF { #2 } = 1
4131     {
4132         \int_set:Nn \l_@@_initial_j_int 1
4133         \bool_set_true:N \l_@@_initial_open_bool
4134     }
4135     {
4136         \cs_if_exist:cTF
4137         {
4138             pgf @ sh @ ns @ \@@_env:
4139             - \int_use:N \l_@@_initial_i_int
4140             - \int_eval:n { #2 - 1 }
4141         }
4142         { \int_set:Nn \l_@@_initial_j_int { #2 - 1 } }
4143         {
4144             \int_set:Nn \l_@@_initial_j_int { #2 }
4145             \bool_set_true:N \l_@@_initial_open_bool
4146         }
4147     }
4148     \int_compare:nNnTF { #2 + #3 - 1 } = \c@jCol
4149     {
4150         \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
4151         \bool_set_true:N \l_@@_final_open_bool
4152     }
4153     {
4154         \cs_if_exist:cTF
4155         {
4156             pgf @ sh @ ns @ \@@_env:
4157             - \int_use:N \l_@@_final_i_int
4158             - \int_eval:n { #2 + #3 }
4159         }
4160         { \int_set:Nn \l_@@_final_j_int { #2 + #3 } }
4161         {
4162             \int_set:Nn \l_@@_final_j_int { #2 + #3 - 1 }
4163             \bool_set_true:N \l_@@_final_open_bool
4164         }
4165     }
4166     \group_begin:

```

```

4167 \int_compare:nNnTF { #1 } = 0
4168 { \color { nicematrix-first-row } }
4169 {
4170   \int_compare:nNnT { #1 } = \g_@@_row_total_int
4171   { \color { nicematrix-last-row } }
4172 }
4173 \keys_set:nn { NiceMatrix / xdots } { #4 }
4174 \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4175 \@@_actually_draw_Ldots:
4176 \group_end:

```

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

```

4177 \int_step_inline:nnn { #2 } { #2 + #3 - 1 }
4178 { \cs_set:cpn { @@ _ dotted _ #1 - ##1 } { } }
4179 }

4180 \hook_gput_code:nnn { begindocument } { . }
4181 {
4182   \tl_set:Nn \l_@@_argspec_tl { 0 { } m 0 { } E { _ ^ } { { } { } } }
4183   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
4184   \exp_args:NNV \NewDocumentCommand \@@_Vdotsfor: \l_@@_argspec_tl
4185   {
4186     \tl_gput_right:Nx \g_@@_HVdotsfor_lines_tl
4187     {
4188       \@@_Vdotsfor:nnnn
4189       { \int_use:N \c@iRow }
4190       { \int_use:N \c@jCol }
4191       { #2 }
4192       {
4193         #1 , #3 ,
4194         down = \exp_not:n { #4 } , up = \exp_not:n { #5 }
4195       }
4196     }
4197   }
4198 }

```

Enf of `\AddToHook`.

```

4199 \cs_new_protected:Npn \@@_Vdotsfor:nnnn #1 #2 #3 #4
4200 {
4201   \bool_set_false:N \l_@@_initial_open_bool
4202   \bool_set_false:N \l_@@_final_open_bool

```

For the column, it’s easy.

```

4203 \int_set:Nn \l_@@_initial_j_int { #2 }
4204 \int_set_eq:NN \l_@@_final_j_int \l_@@_initial_j_int

```

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

```

4205 \int_compare:nNnTF #1 = 1
4206 {
4207   \int_set:Nn \l_@@_initial_i_int 1
4208   \bool_set_true:N \l_@@_initial_open_bool
4209 }
4210 {
4211   \cs_if_exist:cTF
4212   {
4213     pgf @ sh @ ns @ \@@_env:
4214     - \int_eval:n { #1 - 1 }
4215     - \int_use:N \l_@@_initial_j_int
4216   }
4217   { \int_set:Nn \l_@@_initial_i_int { #1 - 1 } }
4218 }

```

```

4219         \int_set:Nn \l_@@_initial_i_int { #1 }
4220         \bool_set_true:N \l_@@_initial_open_bool
4221     }
4222 }
4223 \int_compare:nNnTF { #1 + #3 - 1 } = \c@iRow
4224 {
4225     \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
4226     \bool_set_true:N \l_@@_final_open_bool
4227 }
4228 {
4229     \cs_if_exist:cTF
4230     {
4231         pgf @ sh @ ns @ \@@_env:
4232         - \int_eval:n { #1 + #3 }
4233         - \int_use:N \l_@@_final_j_int
4234     }
4235     { \int_set:Nn \l_@@_final_i_int { #1 + #3 } }
4236     {
4237         \int_set:Nn \l_@@_final_i_int { #1 + #3 - 1 }
4238         \bool_set_true:N \l_@@_final_open_bool
4239     }
4240 }
4241 \group_begin:
4242 \int_compare:nNnTF { #2 } = 0
4243 { \color { nicematrix-first-col } }
4244 {
4245     \int_compare:nNnT { #2 } = \g_@@_col_total_int
4246     { \color { nicematrix-last-col } }
4247 }
4248 \keys_set:nn { NiceMatrix / xdots } { #4 }
4249 \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4250 \@@_actually_draw_Vdots:
4251 \group_end:

```

We declare all the cells concerned by the `\Vdotsfor` as “dotted” (for the dotted lines created by `\Cdots`, `\Ldots`, etc., this job is done by `\@@_find_extremities_of_line:nnnn`). This declaration is done by defining a special control sequence (to nil).

```

4252     \int_step_inline:nnn { #1 } { #1 + #3 - 1 }
4253     { \cs_set:cpn { @@ _ dotted _ ##1 - #2 } { } }
4254 }

```

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

```

4255 \cs_new_protected:Npn \@@_rotate: { \bool_gset_true:N \g_@@_rotate_bool }

```

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.

First, we write a command with an argument of the format i - j and applies the command `\int_eval:n` to i and j ; this must *not* be protected (and is, of course fully expandable).⁷³

```

4256 \cs_new:Npn \@@_double_int_eval:n #1-#2 \q_stop
4257 { \int_eval:n { #1 } - \int_eval:n { #2 } }

```

⁷³Indeed, we want that the user may use the command `\line` in `\CodeAfter` with LaTeX counters in the arguments — with the command `\value`.

With the following construction, the command `\@@_double_int_eval:n` is applied to both arguments before the application of `\@@_line_i:nn` (the construction uses the fact the `\@@_line_i:nn` is protected and that `\@@_double_int_eval:n` is fully expandable).

```

4258 \hook_gput_code:nnn { begindocument } { . }
4259 {
4260   \tl_set:Nn \l_@@_argspec_tl { 0 { } m m ! 0 { } E { _ ^ } { { } { } } }
4261   \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
4262   \exp_args:NNV \NewDocumentCommand \@@_line \l_@@_argspec_tl
4263     {
4264       \group_begin:
4265       \keys_set:nn { NiceMatrix / xdots } { #1 , #4 , down = #5 , up = #6 }
4266       \tl_if_empty:VF \l_@@_xdots_color_tl { \color { \l_@@_xdots_color_tl } }
4267       \use:e
4268       {
4269         \@@_line_i:nn
4270         { \@@_double_int_eval:n #2 \q_stop }
4271         { \@@_double_int_eval:n #3 \q_stop }
4272       }
4273       \group_end:
4274     }
4275 }
4276 \cs_new_protected:Npn \@@_line_i:nn #1 #2
4277 {
4278   \bool_set_false:N \l_@@_initial_open_bool
4279   \bool_set_false:N \l_@@_final_open_bool
4280   \bool_if:nTF
4281     {
4282       \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #1 }
4283       ||
4284       \cs_if_free_p:c { pgf @ sh @ ns @ \@@_env: - #2 }
4285     }
4286     {
4287       \@@_error:nnn { unknown~cell~for~line~in~CodeAfter } { #1 } { #2 }
4288     }
4289     { \@@_draw_line_ii:nn { #1 } { #2 } }
4290 }
4291 \hook_gput_code:nnn { begindocument } { . }
4292 {
4293   \cs_new_protected:Npx \@@_draw_line_ii:nn #1 #2
4294   {

```

We recall that, when externalization is used, `\tikzpicture` and `\endtikzpicture` (or `\pgfpicture` and `\endpgfpicture`) must be directly “visible” and that why we do this static construction of the command `\@@_draw_line_ii:`.

```

4295   \c_@@_pgfortikzpicture_tl
4296   \@@_draw_line_iii:nn { #1 } { #2 }
4297   \c_@@_endpgfortikzpicture_tl
4298 }
4299 }

```

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

```

4300 \cs_new_protected:Npn \@@_draw_line_iii:nn #1 #2
4301 {
4302   \pgfrememberpicturepositiononpagetrue
4303   \pgfpointshapeborder { \@@_env: - #1 } { \@@_qpoint:n { #2 } }
4304   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4305   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4306   \pgfpointshapeborder { \@@_env: - #2 } { \@@_qpoint:n { #1 } }
4307   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4308   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4309   \@@_draw_line:
4310 }

```

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

The command `\RowStyle`

```

4311 \keys_define:nn { NiceMatrix / RowStyle }
4312 {
4313   cell-space-top-limit .dim_set:N = \l_tmpa_dim ,
4314   cell-space-top-limit .initial:n = \c_zero_dim ,
4315   cell-space-top-limit .value_required:n = true ,
4316   cell-space-bottom-limit .dim_set:N = \l_tmpb_dim ,
4317   cell-space-bottom-limit .initial:n = \c_zero_dim ,
4318   cell-space-bottom-limit .value_required:n = true ,
4319   cell-space-limits .meta:n =
4320   {
4321     cell-space-top-limit = #1 ,
4322     cell-space-bottom-limit = #1 ,
4323   } ,
4324   color .tl_set:N = \l_tmpa_tl ,
4325   color .value_required:n = true ,
4326   bold .bool_set:N = \l_tmpa_bool ,
4327   bold .default:n = true ,
4328   bold .initial:n = false ,
4329   nb-rows .int_set:N = \l_@@_key_nb_rows_int ,
4330   nb-rows .value_required:n = true ,
4331   nb-rows .initial:n = 1 ,
4332   rowcolor .tl_set:N = \l_@@_tmpc_tl ,
4333   rowcolor .value_required:n = true ,
4334   rowcolor .initial:n = ,
4335   unknown .code:n = \@@_error:n { Unknown~key-for-RowStyle }
4336 }

```

```

4337 \NewDocumentCommand \@@_RowStyle:n { 0 { } m }
4338 {
4339   \keys_set:nn { NiceMatrix / RowStyle } { #1 }

```

If the key `rowcolor` has been used.

```

4340   \tl_if_empty:NF \l_@@_tmpc_tl
4341   {

```

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

```

4342     \tl_gput_right:Nx \g_nicematrix_code_before_tl
4343     {
4344       \@@_rectanglecolor
4345       { \l_@@_tmpc_tl }
4346       { \int_use:N \c@iRow - \int_use:N \c@jCol }
4347       { \int_use:N \c@iRow - * }
4348     }

```

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

```

4349     \int_compare:nNnT \l_@@_key_nb_rows_int > 1
4350     {
4351       \tl_gput_right:Nx \g_nicematrix_code_before_tl
4352       {
4353         \@@_rowcolor
4354         { \l_@@_tmpc_tl }
4355         {
4356           \int_eval:n { \c@iRow + 1 }
4357           - \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int - 1 }
4358         }
4359       }
4360     }

```



```

4361     }
4362     \tl_gput_right:Nn \g_@@_row_style_tl { \ifnum \c@iRow < }
4363     \tl_gput_right:Nx \g_@@_row_style_tl
4364     { \int_eval:n { \c@iRow + \l_@@_key_nb_rows_int } }
4365     \tl_gput_right:Nn \g_@@_row_style_tl { #2 }
\l_tmpa_dim is the value of the key cell-space-top-limit of \RowStyle.
4366     \dim_compare:nNnT \l_tmpa_dim > \c_zero_dim
4367     {
4368         \tl_gput_right:Nx \g_@@_row_style_tl
4369         {
4370             \tl_gput_right:Nn \exp_not:N \g_@@_post_action_cell_tl
4371             {
4372                 \dim_set:Nn \l_@@_cell_space_top_limit_dim
4373                 { \dim_use:N \l_tmpa_dim }
4374             }
4375         }
4376     }
\l_tmpb_dim is the value of the key cell-space-bottom-limit of \RowStyle.
4377     \dim_compare:nNnT \l_tmpb_dim > \c_zero_dim
4378     {
4379         \tl_gput_right:Nx \g_@@_row_style_tl
4380         {
4381             \tl_gput_right:Nn \exp_not:N \g_@@_post_action_cell_tl
4382             {
4383                 \dim_set:Nn \l_@@_cell_space_bottom_limit_dim
4384                 { \dim_use:N \l_tmpb_dim }
4385             }
4386         }
4387     }
\l_tmpa_tl is the value of the key color of \RowStyle.
4388     \tl_if_empty:NF \l_tmpa_tl
4389     {
4390         \tl_gput_right:Nx \g_@@_row_style_tl
4391         { \mode_leave_vertical: \exp_not:N \color { \l_tmpa_tl } }
4392     }
\l_tmpa_bool is the value of the key bold.
4393     \bool_if:NT \l_tmpa_bool
4394     {
4395         \tl_gput_right:Nn \g_@@_row_style_tl
4396         {
4397             \if_mode_math:
4398                 \c_math_toggle_token
4399                 \bfseries \boldmath
4400                 \c_math_toggle_token
4401             \else:
4402                 \bfseries \boldmath
4403             \fi:
4404         }
4405     }
4406     \tl_gput_right:Nn \g_@@_row_style_tl { \fi }
4407     \g_@@_row_style_tl
4408     \ignorespaces
4409 }

```

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).

```
4410 \cs_new_protected:Npn \@@_add_to_colors_seq:nn #1 #2
4411 {
```

First, we look for the number of the color and, if it's found, we store it in `\l_tmpa_int`. If the color is not present in `\l_@@_colors_seq`, `\l_tmpa_int` will remain equal to 0.

```
4412 \int_zero:N \l_tmpa_int
4413 \seq_map_indexed_inline:Nn \g_@@_colors_seq
4414 { \tl_if_eq:nnT { #1 } { ##2 } { \int_set:Nn \l_tmpa_int { ##1 } } }
4415 \int_compare:nNnTF \l_tmpa_int = \c_zero_int
```

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

```
4416 {
4417   \seq_gput_right:Nn \g_@@_colors_seq { #1 }
4418   \tl_gset:cx { g_@@_color _ \seq_count:N \g_@@_colors_seq _ tl } { #2 }
4419 }
```

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

```
4420 { \tl_gput_right:cx { g_@@_color _ \int_use:N \l_tmpa_int _tl } { #2 } }
4421 }
4422 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { x n }
4423 \cs_generate_variant:Nn \@@_add_to_colors_seq:nn { x x }
```

The macro `\@@_actually_color:` will actually fill all the rectangles, color by color (using the sequence `\l_@@_colors_seq` and all the token lists of the form `\l_@@_color_i_tl`).

```
4424 \cs_new_protected:Npn \@@_actually_color:
4425 {
4426   \pgfpicture
4427   \pgf@relevantforpicturesizefalse
4428   \seq_map_indexed_inline:Nn \g_@@_colors_seq
4429   {
4430     \color ##2
4431     \use:c { g_@@_color _ ##1 _tl }
4432     \tl_gclear:c { g_@@_color _ ##1 _tl }
4433     \pgfusepath { fill }
4434   }
4435   \endpgfpicture
4436 }
4437 \cs_new_protected:Npn \@@_cartesian_color:nn #1 #2
4438 {
4439   \tl_set:Nn \l_@@_rows_tl { #1 }
4440   \tl_set:Nn \l_@@_cols_tl { #2 }
4441   \@@_cartesian_path:
4442 }
```

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

```
4443 \NewDocumentCommand \@@_rowcolor { 0 { } m m }
4444 {
4445   \tl_if_blank:nF { #2 }
4446   {
4447     \@@_add_to_colors_seq:xn
4448     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
4449     { \@@_cartesian_color:nn { #3 } { - } }
4450   }
4451 }
```

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

```
4452 \NewDocumentCommand \@@_columncolor { 0 { } m m }
4453 {
4454   \tl_if_blank:nF { #2 }
4455   {
4456     \@@_add_to_colors_seq:xn
4457     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
4458     { \@@_cartesian_color:nn { - } { #3 } }
4459   }
4460 }
```

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

```
4461 \NewDocumentCommand \@@_rectanglecolor { 0 { } m m m }
4462 {
4463   \tl_if_blank:nF { #2 }
4464   {
4465     \@@_add_to_colors_seq:xn
4466     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
4467     { \@@_rectanglecolor:nnn { #3 } { #4 } { 0 pt } }
4468   }
4469 }
```

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

```
4470 \NewDocumentCommand \@@_roundedrectanglecolor { 0 { } m m m m }
4471 {
4472   \tl_if_blank:nF { #2 }
4473   {
4474     \@@_add_to_colors_seq:xn
4475     { \tl_if_blank:nF { #1 } { [ #1 ] } { #2 } }
4476     { \@@_rectanglecolor:nnn { #3 } { #4 } { #5 } }
4477   }
4478 }
```

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

```
4479 \cs_new_protected:Npn \@@_rectanglecolor:nnn #1 #2 #3
4480 {
4481   \@@_cut_on_hyphen:w #1 \q_stop
4482   \tl_clear_new:N \l_@@_tmpc_tl
4483   \tl_clear_new:N \l_@@_tmpd_tl
4484   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
4485   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
4486   \@@_cut_on_hyphen:w #2 \q_stop
4487   \tl_set:Nx \l_@@_rows_tl { \l_@@_tmpc_tl - \l_tmpa_tl }
4488   \tl_set:Nx \l_@@_cols_tl { \l_@@_tmpd_tl - \l_tmpb_tl }
```

The command `\@@_cartesian_path:n` takes in two implicit arguments: `\l_@@_cols_tl` and `\l_@@_rows_tl`.

```
4489   \@@_cartesian_path:n { #3 }
4490 }
```

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

```

4491 \NewDocumentCommand \@@_cellcolor { 0 { } m m }
4492 {
4493   \clist_map_inline:nn { #3 }
4494     { \@@_rectanglecolor [ #1 ] { #2 } { ##1 } { ##1 } }
4495 }

4496 \NewDocumentCommand \@@_chessboardcolors { 0 { } m m }
4497 {
4498   \int_step_inline:nn { \int_use:N \c@iRow }
4499     {
4500       \int_step_inline:nn { \int_use:N \c@jCol }
4501         {
4502           \int_if_even:nTF { ####1 + ##1 }
4503             { \@@_cellcolor [ #1 ] { #2 } }
4504             { \@@_cellcolor [ #1 ] { #3 } }
4505           { ##1 - ####1 }
4506         }
4507     }
4508 }

```

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”.

```

4509 \NewDocumentCommand \@@_arraycolor { 0 { } m }
4510 {
4511   \@@_rectanglecolor [ #1 ] { #2 }
4512     { 1 - 1 }
4513     { \int_use:N \c@iRow - \int_use:N \c@jCol }
4514 }

4515 \keys_define:nn { NiceMatrix / rowcolors }
4516 {
4517   respect-blocks .bool_set:N = \l_@@_respect_blocks_bool ,
4518   respect-blocks .default:n = true ,
4519   cols .tl_set:N = \l_@@_cols_tl ,
4520   restart .bool_set:N = \l_@@_rowcolors_restart_bool ,
4521   restart .default:n = true ,
4522   unknown .code:n = \@@_error:n { Unknown-key-for-rowcolors }
4523 }

```

The command `\rowcolors` (accessible in the code-before) 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]`.

#1 (optional) is the color space ; **#2** is a list of intervals of rows ; **#3** is the list of colors ; **#4** is for the optional list of pairs *key=value*.

```

4524 \NewDocumentCommand \@@_rowlistcolors { 0 { } m m 0 { } }
4525 {

```

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

```

4526   \group_begin:
4527   \seq_clear_new:N \l_@@_colors_seq
4528   \seq_set_split:Nnn \l_@@_colors_seq { , } { #3 }
4529   \tl_clear_new:N \l_@@_cols_tl
4530   \tl_set:Nn \l_@@_cols_tl { - }
4531   \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).

```

4532   \int_zero_new:N \l_@@_color_int
4533   \int_set:Nn \l_@@_color_int 1
4534   \bool_if:NT \l_@@_respect_blocks_bool
4535   {

```

We don't want to take into account a block which is completely in the “first column” of (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`).

```

4536       \seq_set_eq:NN \l_tmpb_seq \g_@@_pos_of_blocks_seq
4537       \seq_set_filter:Nn \l_tmpa_seq \l_tmpb_seq
4538       { \@@_not_in_exterior_p:nnnnn ##1 }
4539   }
4540   \pgfpicture
4541   \pgf@relevantforpicturesizefalse

```

#2 is the list of intervals of rows.

```

4542   \clist_map_inline:nn { #2 }
4543   {
4544       \tl_set:Nn \l_tmpa_tl { ##1 }
4545       \tl_if_in:NnTF \l_tmpa_tl { - }
4546       { \@@_cut_on_hyphen:w ##1 \q_stop }
4547       { \tl_set:Nx \l_tmpb_tl { \int_use:N \c{iRow } } }

```

Now, `\l_tmpa_tl` and `\l_tmpb_tl` are the first row and the last row of the interval of rows that we have to treat. The counter `\l_tmpa_int` will be the index of the loop over the rows.

```

4548       \int_set:Nn \l_tmpa_int \l_tmpa_tl
4549       \bool_if:NTF \l_@@_rowcolors_restart_bool
4550       { \int_set:Nn \l_@@_color_int 1 }
4551       { \int_set:Nn \l_@@_color_int \l_tmpa_tl }
4552       \int_zero_new:N \l_@@_tmpc_int
4553       \int_set:Nn \l_@@_tmpc_int \l_tmpb_tl
4554       \int_do_until:nNnn \l_tmpa_int > \l_@@_tmpc_int
4555       {

```

We will compute in `\l_tmpb_int` the last row of the “block”.

```

4556           \int_set_eq:NN \l_tmpb_int \l_tmpa_int

```

If the key `respect-blocks` is in force, we have to adjust that value (of course).

```

4557           \bool_if:NT \l_@@_respect_blocks_bool
4558           {
4559               \seq_set_filter:Nn \l_tmpb_seq \l_tmpa_seq
4560               { \@@_intersect_our_row_p:nnnnn ####1 }
4561               \seq_map_inline:Nn \l_tmpb_seq { \@@_rowcolors_i:nnnnn ####1 }

```

Now, the last row of the block is computed in `\l_tmpb_int`.

```

4562           }
4563           \tl_set:Nx \l_@@_rows_tl
4564           { \int_use:N \l_tmpa_int - \int_use:N \l_tmpb_int }

```

`\l_@@_tmpc_tl` will be the color that we will use.

```

4565           \tl_clear_new:N \l_@@_color_tl
4566           \tl_set:Nx \l_@@_color_tl
4567           {
4568               \@@_color_index:n
4569               {
4570                   \int_mod:nn
4571                   { \l_@@_color_int - 1 }
4572                   { \seq_count:N \l_@@_colors_seq }
4573                   + 1
4574               }
4575           }
4576           \tl_if_empty:NF \l_@@_color_tl
4577           {
4578               \@@_add_to_colors_seq:xx

```

```

4579         { \tl_if_blank:nF { #1 } { [ #1 ] } { \l_@@_color_tl } }
4580         { \@@_cartesian_color:nn { \l_@@_rows_tl } { \l_@@_cols_tl } }
4581     }
4582     \int_incr:N \l_@@_color_int
4583     \int_set:Nn \l_tmpa_int { \l_tmpb_int + 1 }
4584 }
4585 }
4586 \endpgfpicture
4587 \group_end:
4588 }

```

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.

```

4589 \cs_new:Npn \@@_color_index:n #1
4590 {
4591     \str_if_eq:eeTF { \seq_item:Nn \l_@@_colors_seq { #1 } } { = }
4592     { \@@_color_index:n { #1 - 1 } }
4593     { \seq_item:Nn \l_@@_colors_seq { #1 } }
4594 }

```

The command `\rowcolors` (available in the `\CodeBefore`) is a specialisation of the most general command `\rowlistcolors`.

```

4595 \NewDocumentCommand \@@_rowcolors { 0 { } m m m 0 { } }
4596 { \@@_rowlistcolors [ #1 ] { #2 } { { #3 } , { #4 } } [ #5 ] }

```

```

4597 \cs_new_protected:Npn \@@_rowcolors_i:nnnnn #1 #2 #3 #4 #5
4598 {
4599     \int_compare:nNnT { #3 } > \l_tmpb_int
4600     { \int_set:Nn \l_tmpb_int { #3 } }
4601 }

```

```

4602 \prg_new_conditional:Nnn \@@_not_in_exterior:nnnnn p
4603 {
4604     \bool_lazy_or:nnTF
4605     { \int_compare_p:nNn { #4 } = \c_zero_int }
4606     { \int_compare_p:nNn { #2 } = { \@@_succ:n { \c_jCol } } }
4607     \prg_return_false:
4608     \prg_return_true:
4609 }

```

The following command return true when the block intersects the row `\l_tmpa_int`.

```

4610 \prg_new_conditional:Nnn \@@_intersect_our_row:nnnnn p
4611 {
4612     \bool_if:nTF
4613     {
4614         \int_compare_p:n { #1 <= \l_tmpa_int }
4615         &&
4616         \int_compare_p:n { \l_tmpa_int <= #3 }
4617     }
4618     \prg_return_true:
4619     \prg_return_false:
4620 }

```

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`).

```

4621 \cs_new_protected:Npn \@@_cartesian_path:n #1
4622 {
4623   \bool_lazy_and:nnT
4624     { ! \seq_if_empty_p:N \l_@@_corners_cells_seq }
4625     { \dim_compare_p:nNn { #1 } = \c_zero_dim }
4626     {
4627       \@@_expand_clist:NN \l_@@_cols_tl \c@jCol
4628       \@@_expand_clist:NN \l_@@_rows_tl \c@iRow
4629     }

```

We begin the loop over the columns.

```

4630   \clist_map_inline:Nn \l_@@_cols_tl
4631   {
4632     \tl_set:Nn \l_tmpa_tl { ##1 }
4633     \tl_if_in:NnTF \l_tmpa_tl { - }
4634       { \@@_cut_on_hyphen:w ##1 \q_stop }
4635       { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
4636     \bool_lazy_or:nnT
4637       { \tl_if_blank_p:V \l_tmpa_tl }
4638       { \str_if_eq_p:Vn \l_tmpa_tl { * } }
4639       { \tl_set:Nn \l_tmpa_tl { 1 } }
4640     \bool_lazy_or:nnT
4641       { \tl_if_blank_p:V \l_tmpb_tl }
4642       { \str_if_eq_p:Vn \l_tmpb_tl { * } }
4643       { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@jCol } }
4644     \int_compare:nNnT \l_tmpb_tl > \c@jCol
4645       { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@jCol } }

```

\l_@@_tmpc_tl will contain the number of column.

```

4646     \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl

```

If we decide to provide the commands \cellcolor, \rectanglecolor, \rowcolor, \columncolor, \rowcolors and \chessboardcolors in the code-before of a \SubMatrix, we will have to modify the following line, by adding a kind of offset. We will have also some other lines to modify.

```

4647     \@@_qpoint:n { col - \l_tmpa_tl }
4648     \int_compare:nNnTF \l_@@_first_col_int = \l_tmpa_tl
4649       { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x - 0.5 \arrayrulewidth } }
4650       { \dim_set:Nn \l_@@_tmpc_dim { \pgf@x + 0.5 \arrayrulewidth } }
4651     \@@_qpoint:n { col - \@@_succ:n \l_tmpb_tl }
4652     \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \arrayrulewidth }

```

We begin the loop over the rows.

```

4653   \clist_map_inline:Nn \l_@@_rows_tl
4654   {
4655     \tl_set:Nn \l_tmpa_tl { #####1 }
4656     \tl_if_in:NnTF \l_tmpa_tl { - }
4657       { \@@_cut_on_hyphen:w #####1 \q_stop }
4658       { \@@_cut_on_hyphen:w #####1 - #####1 \q_stop }
4659     \tl_if_empty:NT \l_tmpa_tl { \tl_set:Nn \l_tmpa_tl { 1 } }
4660     \tl_if_empty:NT \l_tmpb_tl
4661       { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@iRow } }
4662     \int_compare:nNnT \l_tmpb_tl > \c@iRow
4663       { \tl_set:Nx \l_tmpb_tl { \int_use:N \c@iRow } }

```

Now, the numbers of both rows are in \l_tmpa_tl and \l_tmpb_tl.

```

4664     \seq_if_in:NxF \l_@@_corners_cells_seq
4665     { \l_tmpa_tl - \l_@@_tmpc_tl }
4666     {
4667       \@@_qpoint:n { row - \@@_succ:n \l_tmpb_tl }
4668       \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \arrayrulewidth }
4669       \@@_qpoint:n { row - \l_tmpa_tl }
4670       \dim_set:Nn \l_@@_tmpd_dim { \pgf@y + 0.5 \arrayrulewidth }
4671       \pgfsetcornersarced { \pgfpoint { #1 } { #1 } }
4672       \pgfpathrectanglecorners
4673       { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }

```

```

4674         { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
4675     }
4676 }
4677 }
4678 }

```

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`).

```

4679 \cs_new_protected:Npn \@@_cartesian_path: { \@@_cartesian_path:n { 0 pt } }

```

The following command will be used only with `\l_@@_cols_tl` and `\c@jCol` (first case) or with `\l_@@_rows_tl` and `\c@iRow` (second case). For instance, with `\l_@@_cols_tl` equal to 2,4-6,8-* and `\c@jCol` equal to 10, the clist `\l_@@_cols_tl` will be replaced by 2,4,5,6,8,9,10.

```

4680 \cs_new_protected:Npn \@@_expand_clist:NN #1 #2
4681 {
4682   \clist_set_eq:NN \l_tmpa_clist #1
4683   \clist_clear:N #1
4684   \clist_map_inline:Nn \l_tmpa_clist
4685   {
4686     \tl_set:Nn \l_tmpa_tl { ##1 }
4687     \tl_if_in:NnTF \l_tmpa_tl { - }
4688     { \@@_cut_on_hyphen:w ##1 \q_stop }
4689     { \@@_cut_on_hyphen:w ##1 - ##1 \q_stop }
4690     \bool_lazy_or:nnT
4691     { \tl_if_blank_p:V \l_tmpa_tl }
4692     { \str_if_eq_p:Vn \l_tmpa_tl { * } }
4693     { \tl_set:Nn \l_tmpa_tl { 1 } }
4694     \bool_lazy_or:nnT
4695     { \tl_if_blank_p:V \l_tmpb_tl }
4696     { \str_if_eq_p:Vn \l_tmpb_tl { * } }
4697     { \tl_set:Nx \l_tmpb_tl { \int_use:N #2 } }
4698     \int_compare:nNnT \l_tmpb_tl > #2
4699     { \tl_set:Nx \l_tmpb_tl { \int_use:N #2 } }
4700     \int_step_inline:nnn \l_tmpa_tl \l_tmpb_tl
4701     { \clist_put_right:Nn #1 { #####1 } }
4702   }
4703 }

```

When the user uses the key `colortbl-like`, the following command will be linked to `\cellcolor` in the tabular.

```

4704 \NewDocumentCommand \@@_cellcolor_tabular { 0 { } m }
4705 {
4706   \peek_remove_spaces:n
4707   {
4708     \tl_gput_right:Nx \g_nicematrix_code_before_tl
4709     {

```

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

```

4710       \@@_cellcolor [ #1 ] { \exp_not:n { #2 } }
4711       { \int_use:N \c@iRow - \int_use:N \c@jCol }
4712     }
4713   }
4714 }

```

When the user uses the key `colortbl-like`, the following command will be linked to `\rowcolor` in the tabular.

```

4715 \NewDocumentCommand \@@_rowcolor_tabular { 0 { } m }
4716 {
4717   \peek_remove_spaces:n
4718   {

```



```

4719 \tl_gput_right:Nx \g_nicematrix_code_before_tl
4720 {
4721   \@@_rectanglecolor [ #1 ] { \exp_not:n { #2 } }
4722   { \int_use:N \c@iRow - \int_use:N \c@jCol }
4723   { \int_use:N \c@iRow - \exp_not:n { \int_use:N \c@jCol } }
4724 }
4725 }
4726 }

```

```

4727 \NewDocumentCommand \@@_columncolor_preamble { 0 { } m }
4728 {

```

With the following line, we test whether the cell is the first one we encounter in its column (don't forget that some rows may be incomplete).

```

4729 \int_compare:nNnT \c@jCol > \g_@@_col_total_int
4730 {

```

You use `gput_left` because we want the specification of colors for the columns drawn before the specifications of color for the rows (and the cells). Be careful: maybe this is not effective since we have an analyze of the instructions in the `\CodeBefore` in order to fill color by color (to avoid the thin white lines).

```

4731 \tl_gput_left:Nx \g_nicematrix_code_before_tl
4732 {
4733   \exp_not:N \columncolor [ #1 ]
4734   { \exp_not:n { #2 } } { \int_use:N \c@jCol }
4735 }
4736 }
4737 }

```

The vertical and horizontal rules

OnlyMainNiceMatrix

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

We provide the command `\OnlyMainNiceMatrix` in that goal. However, that command must be no-op outside the environments of `nicematrix` (and so the user will be allowed to use the same new type of column in the environments of `nicematrix` and in the standard environments of `array`).

That's why we provide first a global definition of `\OnlyMainNiceMatrix`.

```

4738 \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`.

```

4739 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix:n #1
4740 {
4741   \int_compare:nNnTF \l_@@_first_col_int = 0
4742   { \@@_OnlyMainNiceMatrix_i:n { #1 } }
4743   {
4744     \int_compare:nNnTF \c@jCol = 0
4745     {
4746       \int_compare:nNnF \c@iRow = { -1 }
4747       { \int_compare:nNnF \c@iRow = { \l_@@_last_row_int - 1 } { #1 } }
4748     }
4749     { \@@_OnlyMainNiceMatrix_i:n { #1 } }
4750   }
4751 }

```

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.

```

4752 \cs_new_protected:Npn \@@_OnlyMainNiceMatrix_i:n #1
4753 {
4754   \int_compare:nNnF \c@iRow = 0
4755     { \int_compare:nNnF \c@iRow = \l_@@_last_row_int { #1 } }
4756 }

```

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.

```

4757 \keys_define:nn { NiceMatrix / Rules }
4758 {
4759   position .int_set:N = \l_@@_position_int ,
4760   position .value_required:n = true ,
4761   start .int_set:N = \l_@@_start_int ,
4762   start .initial:n = 1 ,
4763   end .int_set:N = \l_@@_end_int ,

```

The following keys are no-op because there are keys which may be inherited from a list of pairs *key=value* of a definition of a customized rule (with the key `custom-line` of `\NiceMatrixOptions`).

```

4764   letter .code:n = \prg_do_nothing: ,
4765   command .code:n = \prg_do_nothing:
4766 }

```

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

```

4767 \keys_define:nn { NiceMatrix / RulesBis }
4768 {
4769   multiplicity .int_set:N = \l_@@_multiplicity_int ,
4770   multiplicity .initial:n = 1 ,
4771   dotted .bool_set:N = \l_@@_dotted_bool ,
4772   dotted .initial:n = false ,
4773   dotted .default:n = true ,
4774   color .code:n = \@@_set_CT@arc@: #1 \q_stop ,
4775   color .value_required:n = true ,
4776   sep-color .code:n = \@@_set_CT@drsc@: #1 \q_stop ,
4777   sep-color .value_required:n = true ,

```

If the user uses the key `tikz`, the rule (or more precisely: the different sub-rules since a rule may be broken by blocks or others) will be drawn with Tikz.

```

4778   tikz .tl_set:N = \l_@@_tikz_rule_tl ,
4779   tikz .value_required:n = true ,
4780   tikz .initial:n = ,
4781   width .dim_set:N = \l_@@_rule_width_dim ,
4782   width .value_required:n = true
4783 }

```

The vertical rules

The following command will be executed in the internal `\CodeAfter`. The argument `#1` is a list of `key=value` pairs.

```
4784 \cs_new_protected:Npn \@@_vline:n #1
4785 {
```

The group is for the options.

```
4786   \group_begin:
4787   \int_zero_new:N \l_@@_end_int
4788   \int_set_eq:NN \l_@@_end_int \c@iRow
4789   \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).

```
4790   \int_compare:nNnT \l_@@_position_int < { \c@jCol + 2 }
4791     \@@_vline_i:
4792   \group_end:
4793 }
```

```
4794 \cs_new_protected:Npn \@@_vline_i:
4795 {
4796   \int_zero_new:N \l_@@_local_start_int
4797   \int_zero_new:N \l_@@_local_end_int
```

`\l_tmpa_tl` is the number of row and `\l_tmpb_tl` the number of column. When we have found a row corresponding to a rule to draw, we note its number in `\l_@@_tmpc_tl`.

```
4798   \tl_set:Nx \l_tmpb_tl { \int_eval:n \l_@@_position_int }
4799   \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
4800     \l_tmpa_tl
4801   {
```

The boolean `\g_tmpa_bool` indicates whether the small vertical rule will be drawn. If we find that it is in a block (a real block, created by `\Block` or a virtual block corresponding to a dotted line, created by `\Cdots`, `\Vdots`, etc.), we will set `\g_tmpa_bool` to false and the small vertical rule won't be drawn.

```
4802     \bool_gset_true:N \g_tmpa_bool
4803     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
4804       { \@@_test_vline_in_block:nnnnn ##1 }
4805     \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
4806       { \@@_test_vline_in_block:nnnnn ##1 }
4807     \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
4808       { \@@_test_vline_in_stroken_block:nnnn ##1 }
4809     \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_v:
4810     \bool_if:NTF \g_tmpa_bool
4811       {
4812         \int_compare:nNnT \l_@@_local_start_int = 0
```

We keep in memory that we have a rule to draw. `\l_@@_local_start_int` will be the starting row of the rule that we will have to draw.

```
4813         { \int_set:Nn \l_@@_local_start_int \l_tmpa_tl }
4814       }
4815     {
4816       \int_compare:nNnT \l_@@_local_start_int > 0
4817       {
4818         \int_set:Nn \l_@@_local_end_int { \l_tmpa_tl - 1 }
4819         \@@_vline_ii:
4820         \int_zero:N \l_@@_local_start_int
4821       }
4822     }
4823   }
4824   \int_compare:nNnT \l_@@_local_start_int > 0
4825   {
4826     \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
4827     \@@_vline_ii:
```

```

4828     }
4829 }

4830 \cs_new_protected:Npn \@@_test_in_corner_v:
4831 {
4832   \int_compare:nNnTF \l_tmpb_tl = { \@@_succ:n \c@jCol }
4833   {
4834     \seq_if_in:NxT
4835       \l_@@_corners_cells_seq
4836       { \l_tmpa_tl - \@@_pred:n \l_tmpb_tl }
4837       { \bool_set_false:N \g_tmpa_bool }
4838   }
4839   {
4840     \seq_if_in:NxT
4841       \l_@@_corners_cells_seq
4842       { \l_tmpa_tl - \l_tmpb_tl }
4843       {
4844         \int_compare:nNnTF \l_tmpb_tl = 1
4845         { \bool_set_false:N \g_tmpa_bool }
4846         {
4847           \seq_if_in:NxT
4848             \l_@@_corners_cells_seq
4849             { \l_tmpa_tl - \@@_pred:n \l_tmpb_tl }
4850             { \bool_set_false:N \g_tmpa_bool }
4851         }
4852       }
4853   }
4854 }

4855 \cs_new_protected:Npn \@@_vline_ii:
4856 {
4857   \bool_set_false:N \l_@@_dotted_bool
4858   \keys_set:nV { NiceMatrix / RulesBis } \l_@@_other_keys_tl
4859   \bool_if:NTF \l_@@_dotted_bool
4860     \@@_vline_iv:
4861     {
4862       \tl_if_empty:NTF \l_@@_tikz_rule_tl
4863       \@@_vline_iii:
4864       \@@_vline_v:
4865     }
4866 }

```

First the case of a standard rule, that is to say a rule which is not dotted (and the user has not used the key tikz).

```

4867 \cs_new_protected:Npn \@@_vline_iii:
4868 {
4869   \pgfpicture
4870   \pgfrememberpicturepositiononpagetrue
4871   \pgf@relevantforpicturesizefalse
4872   \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
4873   \dim_set_eq:NN \l_tmpa_dim \pgf@y
4874   \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
4875   \dim_set_eq:NN \l_tmpb_dim \pgf@x
4876   \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
4877   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
4878   \bool_lazy_all:nT
4879   {
4880     { \int_compare_p:nNn \l_@@_multiplicity_int > 1 }
4881     { \cs_if_exist_p:N \CT@drsc@ }
4882     { ! \tl_if_blank_p:V \CT@drsc@ }
4883   }

```

```

4884 {
4885   \group_begin:
4886   \CT@drsc@
4887   \dim_add:Nn \l_tmpa_dim { 0.5 \arrayrulewidth }
4888   \dim_sub:Nn \l_@@_tmpc_dim { 0.5 \arrayrulewidth }
4889   \dim_set:Nn \l_@@_tmpd_dim
4890     {
4891       \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
4892       * ( \l_@@_multiplicity_int - 1 )
4893     }
4894   \pgfpathrectanglecorners
4895     { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
4896     { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
4897   \pgfusepath { fill }
4898   \group_end:
4899 }
4900 \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
4901 \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
4902 \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
4903 {
4904   \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
4905   \dim_sub:Nn \l_tmpb_dim \doublerulesep
4906   \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
4907   \pgfpathlineto { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
4908 }
4909 \CT@arc@
4910 \pgfsetlinewidth { 1.1 \arrayrulewidth }
4911 \pgfsetrectcap
4912 \pgfusepathqstroke
4913 \endpgfpicture
4914 }

```

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

```

4915 \cs_new_protected:Npn \@@_vline_iv:
4916 {
4917   \pgfpicture
4918   \pgfrememberpicturepositiononpagetrue
4919   \pgf@relevantforpicturesizefalse
4920   \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
4921   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
4922   \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
4923   \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
4924   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
4925   \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }
4926   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
4927   \CT@arc@
4928   \@@_draw_line:
4929   \endpgfpicture
4930 }

```

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

```

4931 \cs_new_protected:Npn \@@_vline_v:
4932 {
4933   \begin {tikzpicture}
4934   \pgfrememberpicturepositiononpagetrue
4935   \pgf@relevantforpicturesizefalse
4936   \@@_qpoint:n { row - \int_use:N \l_@@_local_start_int }
4937   \dim_set_eq:NN \l_tmpa_dim \pgf@y
4938   \@@_qpoint:n { col - \int_use:N \l_@@_position_int }
4939   \dim_set:Nn \l_tmpb_dim { \pgf@x - 0.5 \l_@@_rule_width_dim }
4940   \@@_qpoint:n { row - \int_eval:n { \l_@@_local_end_int + 1 } }

```

```

4941 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
4942 \exp_args:N \tikzset \l_@@_tikz_rule_tl
4943 \use:x { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
4944 ( \l_tmpb_dim , \l_tmpa_dim ) --
4945 ( \l_tmpb_dim , \l_@@_tmpc_dim ) ;
4946 \end { tikzpicture }
4947 }

```

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).

```

4948 \cs_new_protected:Npn \@@_draw_vlines:
4949 {
4950   \int_step_inline:nnn
4951   {
4952     \bool_if:NTF { \l_@@_NiceArray_bool && ! \l_@@_except_borders_bool }
4953     1 2
4954   }
4955   {
4956     \bool_if:NTF { \l_@@_NiceArray_bool && ! \l_@@_except_borders_bool }
4957     { \@@_succ:n \c@jCol }
4958     \c@jCol
4959   }
4960   {
4961     \tl_if_eq:NnF \l_@@_vlines_clist { all }
4962     { \clist_if_in:NnT \l_@@_vlines_clist { ##1 } }
4963     { \@@_vline:n { position = ##1 } }
4964   }
4965 }

```

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}`.

```

4966 \cs_new_protected:Npn \@@_hline:n #1
4967 {

```

The group is for the options.

```

4968   \group_begin:
4969   \int_zero_new:N \l_@@_end_int
4970   \int_set_eq:NN \l_@@_end_int \c@jCol
4971   \keys_set_known:nnN { NiceMatrix / Rules } { #1 } \l_@@_other_keys_tl
4972   \@@_hline_i:
4973   \group_end:
4974 }

```

```

4975 \cs_new_protected:Npn \@@_hline_i:
4976 {
4977   \int_zero_new:N \l_@@_local_start_int
4978   \int_zero_new:N \l_@@_local_end_int

```

`\l_tmpa_tl` is the number of row and `\l_tmpb_tl` the number of column. When we have found a column corresponding to a rule to draw, we note its number in `\l_@@_tmpc_tl`.

```

4979   \tl_set:Nx \l_tmpa_tl { \int_use:N \l_@@_position_int }
4980   \int_step_variable:nnNn \l_@@_start_int \l_@@_end_int
4981   \l_tmpb_tl
4982   {

```

The boolean `\g_tmpa_bool` indicates whether the small horizontal rule will be drawn. If we find that it is in a block (a real block, created by `\Block` or a virtual block corresponding to a dotted line, created by `\Cdots`, `\Vdots`, etc.), we will set `\g_tmpa_bool` to `false` and the small horizontal rule won't be drawn.

```

4983     \bool_gset_true:N \g_tmpa_bool
4984     \seq_map_inline:Nn \g_@@_pos_of_blocks_seq

```

```

4985     { \@@_test_hline_in_block:nnnnn ##1 }
4986 \seq_map_inline:Nn \g_@@_pos_of_xdots_seq
4987     { \@@_test_hline_in_block:nnnnn ##1 }
4988 \seq_map_inline:Nn \g_@@_pos_of_stroken_blocks_seq
4989     { \@@_test_hline_in_stroken_block:nnnn ##1 }
4990 \clist_if_empty:NF \l_@@_corners_clist \@@_test_in_corner_h:
4991 \bool_if:NTF \g_tmpa_bool
4992     {
4993     \int_compare:nNnT \l_@@_local_start_int = 0

```

We keep in memory that we have a rule to draw. \l_@@_local_start_int will be the starting row of the rule that we will have to draw.

```

4994         { \int_set:Nn \l_@@_local_start_int \l_tmpb_tl }
4995     }
4996     {
4997     \int_compare:nNnT \l_@@_local_start_int > 0
4998     {
4999     \int_set:Nn \l_@@_local_end_int { \l_tmpb_tl - 1 }
5000     \@@_hline_ii:
5001     \int_zero:N \l_@@_local_start_int
5002     }
5003     }
5004 }
5005 \int_compare:nNnT \l_@@_local_start_int > 0
5006 {
5007     \int_set_eq:NN \l_@@_local_end_int \l_@@_end_int
5008     \@@_hline_ii:
5009 }
5010 }

```

```

5011 \cs_new_protected:Npn \@@_test_in_corner_h:
5012 {
5013     \int_compare:nNnTF \l_tmpa_tl = { \@@_succ:n \c@iRow }
5014     {
5015     \seq_if_in:NxT
5016     \l_@@_corners_cells_seq
5017     { \@@_pred:n \l_tmpa_tl - \l_tmpb_tl }
5018     { \bool_set_false:N \g_tmpa_bool }
5019     }
5020     {
5021     \seq_if_in:NxT
5022     \l_@@_corners_cells_seq
5023     { \l_tmpa_tl - \l_tmpb_tl }
5024     {
5025     \int_compare:nNnTF \l_tmpa_tl = 1
5026     { \bool_set_false:N \g_tmpa_bool }
5027     {
5028     \seq_if_in:NxT
5029     \l_@@_corners_cells_seq
5030     { \@@_pred:n \l_tmpa_tl - \l_tmpb_tl }
5031     { \bool_set_false:N \g_tmpa_bool }
5032     }
5033     }
5034 }
5035 }

```

```

5036 \cs_new_protected:Npn \@@_hline_ii:
5037 {
5038     \bool_set_false:N \l_@@_dotted_bool
5039     \keys_set:nV { NiceMatrix / RulesBis } \l_@@_other_keys_tl
5040     \bool_if:NTF \l_@@_dotted_bool
5041     \@@_hline_iv:

```

```

5042     {
5043         \tl_if_empty:NTF \l_@@_tikz_rule_tl
5044             \@@_hline_iii:
5045             \@@_hline_v:
5046     }
5047 }

```

First the case of a standard rule, that is to say a rule which is not dotted.

```

5048 \cs_new_protected:Npn \@@_hline_iii:
5049 {
5050     \pgfpicture
5051     \pgfrememberpicturepositiononpagetrue
5052     \pgf@relevantforpicturesizefalse
5053     \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
5054     \dim_set_eq:NN \l_tmpa_dim \pgf@x
5055     \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
5056     \dim_set_eq:NN \l_tmpb_dim \pgf@y
5057     \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
5058     \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
5059     \bool_lazy_all:nT
5060     {
5061         { \int_compare_p:nNn \l_@@_multiplicity_int > 1 }
5062         { \cs_if_exist_p:N \CT@drsc@ }
5063         { ! \tl_if_blank_p:V \CT@drsc@ }
5064     }
5065     {
5066         \group_begin:
5067         \CT@drsc@
5068         \dim_set:Nn \l_@@_tmpd_dim
5069         {
5070             \l_tmpb_dim - ( \doublerulesep + \arrayrulewidth )
5071             * ( \l_@@_multiplicity_int - 1 )
5072         }
5073         \pgfpathrectanglecorners
5074         { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5075         { \pgfpoint \l_@@_tmpc_dim \l_@@_tmpd_dim }
5076         \pgfusepathqfill
5077         \group_end:
5078     }
5079     \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5080     \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
5081     \prg_replicate:nn { \l_@@_multiplicity_int - 1 }
5082     {
5083         \dim_sub:Nn \l_tmpb_dim \arrayrulewidth
5084         \dim_sub:Nn \l_tmpb_dim \doublerulesep
5085         \pgfpathmoveto { \pgfpoint \l_tmpa_dim \l_tmpb_dim }
5086         \pgfpathlineto { \pgfpoint \l_@@_tmpc_dim \l_tmpb_dim }
5087     }
5088     \CT@arc@
5089     \pgfsetlinewidth { 1.1 \arrayrulewidth }
5090     \pgfsetrectcap
5091     \pgfusepathqstroke
5092     \endpgfpicture
5093 }

```

The following code is for the case of a dotted rule (with our system of rounded dots). The aim is that, by standard the dotted line fits between square brackets (\hline doesn't).


```

\begin{bNiceMatrix}
1 & 2 & 3 & 4 \\
\hline
1 & 2 & 3 & 4 \\
\hdottedline
1 & 2 & 3 & 4
\end{bNiceMatrix}

```

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ \hdots & \hdots & \hdots & \hdots \\ 1 & 2 & 3 & 4 \end{bmatrix}$$

But, if the user uses `margin`, the dotted line extends to have the same width as a `\hline`.

```

\begin{bNiceMatrix}[margin]
1 & 2 & 3 & 4 \\
\hline
1 & 2 & 3 & 4 \\
\hdottedline
1 & 2 & 3 & 4
\end{bNiceMatrix}

```

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ \hdots & \hdots & \hdots & \hdots \\ 1 & 2 & 3 & 4 \end{bmatrix}$$

```

5094 \cs_new_protected:Npn \@@_hline_iv:
5095 {
5096   \pgfpicture
5097   \pgfrememberpicturepositiononpagetrue
5098   \pgf@relevantforpicturesizefalse
5099   \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
5100   \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
5101   \dim_set_eq:NN \l_@@_y_final_dim \pgf@y
5102   \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
5103   \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
5104   \int_compare:nNnT \l_@@_local_start_int = 1
5105   {
5106     \dim_sub:Nn \l_@@_x_initial_dim \l_@@_left_margin_dim
5107     \bool_if:NT \l_@@_NiceArray_bool
5108     { \dim_sub:Nn \l_@@_x_initial_dim \arraycolsep }

```

For reasons purely aesthetic, we do an adjustment in the case of a rounded bracket. The correction by `0.5 \l_@@_inter_dots_dim` is *ad hoc* for a better result.

```

5109     \tl_if_eq:NnF \g_@@_left_delim_tl (
5110       { \dim_add:Nn \l_@@_x_initial_dim { 0.5 \l_@@_inter_dots_dim } }
5111     )
5112     \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
5113     \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
5114     \int_compare:nNnT \l_@@_local_end_int = \c@jCol
5115     {
5116       \dim_add:Nn \l_@@_x_final_dim \l_@@_right_margin_dim
5117       \bool_if:NT \l_@@_NiceArray_bool
5118       { \dim_add:Nn \l_@@_x_final_dim \arraycolsep }
5119       \tl_if_eq:NnF \g_@@_right_delim_tl )
5120       { \dim_gsub:Nn \l_@@_x_final_dim { 0.5 \l_@@_inter_dots_dim } }
5121     }
5122     \CT@arc@
5123     \@@_draw_line:
5124     \endpgfpicture
5125   }

```

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

```

5126 \cs_new_protected:Npn \@@_hline_v:
5127 {
5128   \begin { tikzpicture }
5129   \pgfrememberpicturepositiononpagetrue
5130   \pgf@relevantforpicturesizefalse
5131   \@@_qpoint:n { col - \int_use:N \l_@@_local_start_int }
5132   \dim_set_eq:NN \l_tmpa_dim \pgf@x
5133   \@@_qpoint:n { row - \int_use:N \l_@@_position_int }
5134   \dim_set:Nn \l_tmpb_dim { \pgf@y - 0.5 \l_@@_rule_width_dim }

```

```

5135 \@@_qpoint:n { col - \int_eval:n { \l_@@_local_end_int + 1 } }
5136 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@x
5137 \exp_args:NV \tikzset \l_@@_tikz_rule_tl
5138 \use:x { \exp_not:N \draw [ \l_@@_tikz_rule_tl ] }
5139 ( \l_tmpa_dim , \l_tmpb_dim ) --
5140 ( \l_@@_tmpc_dim , \l_tmpb_dim ) ;
5141 \end { tikzpicture }
5142 }

```

The command `\@@_draw_hlines:` draws all the horizontal rules excepted in the blocks (even the virtual blocks determined by commands such as `\Cdots` and in the corners (if the key `corners` is used)).

```

5143 \cs_new_protected:Npn \@@_draw_hlines:
5144 {
5145   \int_step_inline:nnn
5146   {
5147     \bool_if:NTF { \l_@@_NiceArray_bool && ! \l_@@_except_borders_bool }
5148     1 2
5149   }
5150   {
5151     \bool_if:NTF { \l_@@_NiceArray_bool && ! \l_@@_except_borders_bool }
5152     { \@@_succ:n \c@iRow }
5153     \c@iRow
5154   }
5155   {
5156     \tl_if_eq:NnF \l_@@_hlines_clist { all }
5157     { \clist_if_in:NnT \l_@@_hlines_clist { ##1 } }
5158     { \@@_hline:n { position = ##1 } }
5159   }
5160 }

```

The command `\@@_Hline:` will be linked to `\Hline` in the environments of `nicematrix`.

```

5161 \cs_set:Npn \@@_Hline: { \noalign { \ifnum 0 = ` } \fi \@@_Hline_i:n { 1 } }

```

The argument of the command `\@@_Hline_i:n` is the number of successive `\Hline` found.

```

5162 \cs_set:Npn \@@_Hline_i:n #1
5163 {
5164   \peek_meaning_ignore_spaces:NTF \Hline
5165   { \@@_Hline_ii:nn { #1 + 1 } }
5166   { \@@_Hline_iii:n { #1 } }
5167 }
5168 \cs_set:Npn \@@_Hline_ii:nn #1 #2 { \@@_Hline_i:n { #1 } }
5169 \cs_set:Npn \@@_Hline_iii:n #1
5170 {
5171   \skip_vertical:n
5172   {
5173     \arrayrulewidth * ( #1 )
5174     + \doublerulesep * ( \int_max:nn 0 { #1 - 1 } )
5175   }
5176   \tl_gput_right:Nx \g_@@_internal_code_after_tl
5177   {
5178     \@@_hline:n
5179     {
5180       position = \int_eval:n { \c@iRow + 1 } ,
5181       multiplicity = #1
5182     }
5183   }
5184   \ifnum 0 = ` { \fi }
5185 }

```

Customized rules defined by the final user

The final user can define a customized rule by using the key `custom-line` in `\NiceMatrixOptions`. That key takes in as value a list of `key=value` pairs.

Among the keys available in that list, there is the key `letter` to specify a letter that the final user will use in the preamble of the array. All the letters defined by this way by the final user for such customized rules are added in the set of keys `{NiceMatrix / ColumnTypes}`. That set of keys is used to store the characteristics of those types of rules for convenience: the keys of that set of keys won't never be used as keys by the final user (he will use, instead, letters in the preamble of its array).

```
5186 \keys_define:nn { NiceMatrix / ColumnTypes } { }
```

The following command will create the customized rule (it is executed when the final user uses the key `custom-line` in `\NiceMatrixOptions`).

```
5187 \cs_new_protected:Npn \@@_custom_line:n #1
5188 {
5189   \str_clear_new:N \l_@@_command_str
5190   \str_clear_new:N \l_@@_letter_str
5191   \dim_zero_new:N \l_@@_rule_width_dim
```

The token list `\l_tmpa_tl` is for the key `color`.

```
5192   \tl_clear:N \l_tmpa_tl
```

The flag `\l_tmpa_bool` will indicate whether the key `tikz` is present.

```
5193   \bool_set_false:N \l_tmpa_bool
```

The flag `\l_tmpb_bool` will indicate whether the key `width` is present.

```
5194   \bool_set_false:N \l_tmpb_bool
5195   \keys_set_known:nn { NiceMatrix / Custom-Line } { #1 }
5196   \bool_if:NT \l_tmpa_bool
5197   {
```

We can't use `\c_@@_tikz_loaded_bool` to test whether `tikz` is loaded because `\NiceMatrixOptions` may be used in the preamble of the document.

```
5198     \cs_if_exist:NF \tikzpicture
5199     { \@@_error:n { tikz-in-custom-line-without-tikz } }
5200     \tl_if_empty:NF \l_tmpa_tl
5201     { \@@_error:n { color-in-custom-line-with-tikz } }
5202   }
5203   \bool_if:NT \l_tmpb_bool
5204   {
5205     \bool_if:NF \l_tmpa_bool
5206     { \@@_error:n { key-width-without-key-tikz } }
5207   }
```

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.

```
5208   \bool_lazy_and:nnTF
5209   { \str_if_empty_p:N \l_@@_letter_str }
5210   { \str_if_empty_p:N \l_@@_command_str }
5211   { \@@_error:n { No-letter-and-no-command } }
5212   {
5213     \str_if_empty:NF \l_@@_letter_str
5214     {
5215       \int_compare:nNnTF { \str_count:N \l_@@_letter_str } = 1
5216       {
5217         \exp_args:NnV \tl_if_in:NnTF
5218         \c_@@_forbidden_letters_str \l_@@_letter_str
5219         { \@@_error:n { Forbidden-letter } }
5220       }
5221       \exp_args:Nnx \keys_define:nn { NiceMatrix / ColumnTypes }
5222       {
```

```

5223         \l_@@_letter_str .code:n =
5224         { \@@_custom_line_i:n { \exp_not:n { #1 } } }
5225     }
5226 }
5227 }
5228 { \@@_error:n { Several-letters } }
5229 }
5230 \str_if_empty:NF \l_@@_command_str
5231 {

```

The flag `\l_tmpa_bool` means that the key 'tikz' have been used. When the key 'tikz' has not been used, the width of the rule is computed with the multiplicity of the rule.

```

5232     \bool_if:NF \l_tmpa_bool
5233     {
5234         \dim_set:Nn \l_@@_rule_width_dim
5235         {
5236             \arrayrulewidth * \l_@@_tmpc_int
5237             + \doublerulesep * ( \l_@@_tmpc_int - 1 )
5238         }
5239     }
5240     \exp_args:NnV \@@_define_h_custom_line:nn
5241     { #1 }
5242     \l_@@_rule_width_dim
5243 }
5244 }
5245 }
5246 \str_const:Nn \c_@@_forbidden_letters_str { lcrpmbVX:|()!@<> }

```

The previous command `\@@_custom_line:n` uses the following set of keys. However, the whole definition of the customized lines (as provided by the final user as argument of `custom-line`) will also be used further with other sets of keys (for instance `{NiceMatrix/Rules}`). That's why the following set of keys has only entries for a few keys.

```

5247 \keys_define:nn { NiceMatrix / Custom-Line }
5248 {
5249     letter .str_set:N = \l_@@_letter_str ,
5250     letter .value_required:n = true ,
5251     command .str_set:N = \l_@@_command_str ,
5252     command .value_required:n = true ,
5253     multiplicity .int_set:N = \l_@@_tmpc_int ,
5254     multiplicity .initial:n = 1 ,
5255     multiplicity .value_required:n = true ,
5256     color .tl_set:N = \l_tmpa_tl ,
5257     color .value_required:n = true ,

```

When the key `tikz` is used, the rule will be drawn with Tikz by using the set of keys specified in the value of that key `tikz`.

```

5258     tikz .code:n = \bool_set_true:N \l_tmpa_bool ,

```

The key `width` must be used only when the key `tikz` is used. When used, the key `width` specifies the width of the rule: it will be used to reserve space (in the preamble of the array or in the command for the horizontal rules).

```

5259     width .code:n = \dim_set:Nn \l_@@_rule_width_dim { #1 }
5260                 \bool_set_true:N \l_tmpb_bool ,
5261     width .value_required:n = true ,
5262 }

```

The following command will create the command that the final user will use in its array to draw a horizontal rule (hence the 'h' in the name). `#1` is the whole set of keys to pass to `\@@_line:n` and `#2` is the width of the whole rule.

```

5263 \cs_new_protected:Npn \@@_define_h_custom_line:nn #1 #2
5264 {

```

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`.

```

5265 \cs_set:cpn \l_@@_command_str
5266 {
5267   \noalign
5268   {
5269     \skip_vertical:n { #2 }
5270     \tl_gput_right:Nx \g_@@_internal_code_after_tl
5271     { \@@_hline:n { #1 , position = \int_eval:n { \c@iRow + 1 } } }
5272   }
5273 }
5274 }

```

The flag `\l_tmpa_bool` means that the key 'tikz' have been used. When the key 'tikz' has not been used, the width of the rule is computed with the multiplicity of the rule.

```

5275 \cs_new_protected:Npn \@@_custom_line_i:n #1
5276 {
5277   \bool_if:NF \l_tmpa_bool
5278   {
5279     \dim_set:Nn \l_@@_rule_width_dim
5280     {
5281       \arrayrulewidth * \l_@@_tmpc_int
5282       + \doublerulesep * ( \l_@@_tmpc_int - 1 )
5283     }
5284   }
5285   \tl_gput_right:Nx \g_@@_preamble_tl
5286   {
5287     \exp_not:N !
5288     { \skip_horizontal:n { \dim_use:N \l_@@_rule_width_dim } }
5289   }
5290   \tl_gput_right:Nx \g_@@_internal_code_after_tl
5291   { \@@_vline:n { #1 , position = \@@_succ:n \c@jCol } }
5292 }

```

The key hvlines

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

```

5293 \cs_new_protected:Npn \@@_test_hline_in_block:nnnnn #1 #2 #3 #4
5294 {
5295   \bool_lazy_all:nT
5296   {
5297     { \int_compare_p:nNn \l_tmpa_tl > { #1 } }
5298     { \int_compare_p:nNn \l_tmpa_tl < { #3 + 1 } }
5299     { \int_compare_p:nNn \l_tmpb_tl > { #2 - 1 } }
5300     { \int_compare_p:nNn \l_tmpb_tl < { #4 + 1 } }
5301   }
5302   { \bool_gset_false:N \g_tmpa_bool }
5303 }

```

The same for vertical rules.

```

5304 \cs_new_protected:Npn \@@_test_vline_in_block:nnnnn #1 #2 #3 #4
5305 {
5306   \bool_lazy_all:nT
5307   {
5308     { \int_compare_p:nNn \l_tmpa_tl > { #1 - 1 } }
5309     { \int_compare_p:nNn \l_tmpa_tl < { #3 + 1 } }
5310     { \int_compare_p:nNn \l_tmpb_tl > { #2 } }
5311     { \int_compare_p:nNn \l_tmpb_tl < { #4 + 1 } }
5312   }
5313   { \bool_gset_false:N \g_tmpa_bool }
5314 }

```

```

5315 \cs_new_protected:Npn \@@_test_hline_in_stroken_block:nnnn #1 #2 #3 #4
5316 {
5317   \bool_lazy_all:nT
5318   {
5319     { \int_compare_p:nNn \l_tmpa_tl > { #1 - 1 } }
5320     { \int_compare_p:nNn \l_tmpa_tl < { #3 + 2 } }
5321     { \int_compare_p:nNn \l_tmpb_tl > { #2 - 1 } }
5322     { \int_compare_p:nNn \l_tmpb_tl < { #4 + 1 } }
5323   }
5324   { \bool_gset_false:N \g_tmpa_bool }
5325 }
5326 \cs_new_protected:Npn \@@_test_vline_in_stroken_block:nnnn #1 #2 #3 #4
5327 {
5328   \bool_lazy_all:nT
5329   {
5330     { \int_compare_p:nNn \l_tmpa_tl > { #1 - 1 } }
5331     { \int_compare_p:nNn \l_tmpa_tl < { #3 + 1 } }
5332     { \int_compare_p:nNn \l_tmpb_tl > { #2 - 1 } }
5333     { \int_compare_p:nNn \l_tmpb_tl < { #4 + 2 } }
5334   }
5335   { \bool_gset_false:N \g_tmpa_bool }
5336 }

```

The key corners

When the `key corners` is raised, the rules are not drawn in the corners. Of course, we have to compute the corners before we begin to draw the rules.

```

5337 \cs_new_protected:Npn \@@_compute_corners:
5338 {

```

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.

```

5339   \seq_clear_new:N \l_@@_corners_cells_seq
5340   \clist_map_inline:Nn \l_@@_corners_clist
5341   {
5342     \str_case:nnF { ##1 }
5343     {
5344       { NW }
5345       { \@@_compute_a_corner:nnnnnn 1 1 1 1 \c@iRow \c@jCol }
5346       { NE }
5347       { \@@_compute_a_corner:nnnnnn 1 \c@jCol 1 { -1 } \c@iRow 1 }
5348       { SW }
5349       { \@@_compute_a_corner:nnnnnn \c@iRow 1 { -1 } 1 1 \c@jCol }
5350       { SE }
5351       { \@@_compute_a_corner:nnnnnn \c@iRow \c@jCol { -1 } { -1 } 1 1 }
5352     }
5353     { \@@_error:nn { bad~corner } { ##1 } }
5354   }

```

Even if the user has used the `key corners` (or the `key hvlines-except-corners`), the list of cells in the corners may be empty.

```

5355   \seq_if_empty:NF \l_@@_corners_cells_seq
5356   {

```

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.

```

5357     \tl_gput_right:Nx \g_@@_aux_tl
5358     {
5359       \seq_set_from_clist:Nn \exp_not:N \l_@@_corners_cells_seq
5360       { \seq_use:Nnnn \l_@@_corners_cells_seq , , , }

```

```

5361     }
5362   }
5363 }

```

“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.

```

5364 \cs_new_protected:Npn \@@_compute_a_corner:nnnnnn #1 #2 #3 #4 #5 #6
5365 {

```

For the explanations and the name of the variables, we consider that we are computing the left-upper corner.

First, we try to determine which is the last empty cell (and not in a block: we won’t add that precision any longer) in the column of number 1. The flag `\l_tmpa_bool` will be raised when a non-empty cell is found.

```

5366   \bool_set_false:N \l_tmpa_bool
5367   \int_zero_new:N \l_@@_last_empty_row_int
5368   \int_set:Nn \l_@@_last_empty_row_int { #1 }
5369   \int_step_inline:nnnn { #1 } { #3 } { #5 }
5370   {
5371     \@@_test_if_cell_in_a_block:nn { ##1 } { \int_eval:n { #2 } }
5372     \bool_lazy_or:nnTF
5373     {
5374       \cs_if_exist_p:c
5375       { pgf @ sh @ ns @ \@@_env: - ##1 - \int_eval:n { #2 } }
5376     }
5377     \l_tmpb_bool
5378     { \bool_set_true:N \l_tmpa_bool }
5379     {
5380       \bool_if:NF \l_tmpa_bool
5381       { \int_set:Nn \l_@@_last_empty_row_int { ##1 } }
5382     }
5383   }

```

Now, you determine the last empty cell in the row of number 1.

```

5384   \bool_set_false:N \l_tmpa_bool
5385   \int_zero_new:N \l_@@_last_empty_column_int
5386   \int_set:Nn \l_@@_last_empty_column_int { #2 }
5387   \int_step_inline:nnnn { #2 } { #4 } { #6 }
5388   {
5389     \@@_test_if_cell_in_a_block:nn { \int_eval:n { #1 } } { ##1 }
5390     \bool_lazy_or:nnTF
5391     \l_tmpb_bool
5392     {
5393       \cs_if_exist_p:c
5394       { pgf @ sh @ ns @ \@@_env: - \int_eval:n { #1 } - ##1 }
5395     }
5396     { \bool_set_true:N \l_tmpa_bool }
5397     {
5398       \bool_if:NF \l_tmpa_bool
5399       { \int_set:Nn \l_@@_last_empty_column_int { ##1 } }
5400     }
5401   }

```

Now, we loop over the rows.

```
5402 \int_step_inline:nnnn { #1 } { #3 } \l_@@_last_empty_row_int
5403 {
```

We treat the row number ##1 with another loop.

```
5404 \bool_set_false:N \l_tmpa_bool
5405 \int_step_inline:nnnn { #2 } { #4 } \l_@@_last_empty_column_int
5406 {
5407   \@@_test_if_cell_in_a_block:nn { ##1 } { #####1 }
5408   \bool_lazy_or:nnTF
5409     \l_tmpb_bool
5410     {
5411       \cs_if_exist_p:c
5412         { pgf @ sh @ ns @ \@@_env: - ##1 - #####1 }
5413     }
5414   { \bool_set_true:N \l_tmpa_bool }
5415   {
5416     \bool_if:NF \l_tmpa_bool
5417     {
5418       \int_set:Nn \l_@@_last_empty_column_int { #####1 }
5419       \seq_put_right:Nn
5420         \l_@@_corners_cells_seq
5421         { ##1 - #####1 }
5422     }
5423   }
5424 }
5425 }
5426 }
```

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`).

```
5427 \cs_new_protected:Npn \@@_test_if_cell_in_a_block:nn #1 #2
5428 {
5429   \int_set:Nn \l_tmpa_int { #1 }
5430   \int_set:Nn \l_tmpb_int { #2 }
5431   \bool_set_false:N \l_tmpb_bool
5432   \seq_map_inline:Nn \g_@@_pos_of_blocks_seq
5433     { \@@_test_if_cell_in_block:nnnnnnn \l_tmpa_int \l_tmpb_int ##1 }
5434 }
5435 \cs_new_protected:Npn \@@_test_if_cell_in_block:nnnnnnn #1 #2 #3 #4 #5 #6 #7
5436 {
5437   \int_compare:nNnT { #3 } < { \@@_succ:n { #1 } }
5438   {
5439     \int_compare:nNnT { #1 } < { \@@_succ:n { #5 } }
5440     {
5441       \int_compare:nNnT { #4 } < { \@@_succ:n { #2 } }
5442       {
5443         \int_compare:nNnT { #2 } < { \@@_succ:n { #6 } }
5444         { \bool_set_true:N \l_tmpb_bool }
5445       }
5446     }
5447   }
5448 }
```

The commands to draw dotted lines to separate columns and rows

These commands don't use the normal nodes, the medium nor the large nodes. They only use the `col` nodes and the `row` nodes.

Horizontal dotted lines

The following command must *not* be protected because it's meant to be expanded in a `\noalign`.

```
5449 \cs_new:Npn \@@_hdottedline:
5450 {
5451   \noalign { \skip_vertical:N 2\l_@@_radius_dim }
5452   \@@_hdottedline_i:
5453 }
```

On the other side, the following command should be protected.

```
5454 \cs_new_protected:Npn \@@_hdottedline_i:
5455 {
```

We write in the internal `\CodeAfter` the instruction that will actually draw the dotted line. It's not possible to draw this dotted line now because we don't know the length of the line (we don't even know the number of columns).

```
5456   \tl_gput_right:Nx \g_@@_internal_code_after_tl
5457   { \@@_hdottedline:n { \int_use:N \c@iRow } }
5458 }
```

The command `\@@_hdottedline:n` is the command written in the internal `\CodeAfter` that will actually draw the dotted line. Its argument is the number of the row *before* which we will draw the row.

```
5459 \cs_new_protected:Npn \@@_hdottedline:n #1
5460 { \@@_hline:n { position = #1 , end = \int_use:N \c@jCol , dotted } }
```

Vertical dotted lines

```
5461 \cs_new_protected:Npn \@@_vdottedline:n #1
5462 { \@@_vline:n { position = \int_eval:n { #1 + 1 } , dotted } }
```

The environment {NiceMatrixBlock}

The following flag will be raised when all the columns of the environments of the block must have the same width in “auto” mode.

```
5463 \bool_new:N \l_@@_block_auto_columns_width_bool
```

Up to now, there is only one option available for the environment `{NiceMatrixBlock}`.

```
5464 \keys_define:nn { NiceMatrix / NiceMatrixBlock }
5465 {
5466   auto-columns-width .code:n =
5467   {
5468     \bool_set_true:N \l_@@_block_auto_columns_width_bool
5469     \dim_gzero_new:N \g_@@_max_cell_width_dim
5470     \bool_set_true:N \l_@@_auto_columns_width_bool
5471   }
5472 }

5473 \NewDocumentEnvironment { NiceMatrixBlock } { ! O { } }
5474 {
5475   \int_gincr:N \g_@@_NiceMatrixBlock_int
5476   \dim_zero:N \l_@@_columns_width_dim
5477   \keys_set:nn { NiceMatrix / NiceMatrixBlock } { #1 }
5478   \bool_if:NT \l_@@_block_auto_columns_width_bool
5479   {
5480     \cs_if_exist:cT { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
5481     {
5482       \exp_args:NNc \dim_set:Nn \l_@@_columns_width_dim
5483       { @@_max_cell_width_ \int_use:N \g_@@_NiceMatrixBlock_int }
5484     }
5485   }
5486 }
```

At the end of the environment `{NiceMatrixBlock}`, we write in the main `aux` file instructions for the column width of all the environments of the block (that's why we have stored the number of the first environment of the block in the counter `\l_@@_first_env_block_int`).

```

5487 {
5488   \bool_if:NT \l_@@_block_auto_columns_width_bool
5489   {
5490     \iow_shipout:Nn \@mainaux \ExplSyntaxOn
5491     \iow_shipout:Nx \@mainaux
5492     {
5493       \cs_gset:cpn
5494       { @@ _ max _ cell _ width _ \int_use:N \g_@@_NiceMatrixBlock_int }
5495       { \dim_eval:n { \g_@@_max_cell_width_dim + \arrayrulewidth } }
5496     }
5497     \iow_shipout:Nn \@mainaux \ExplSyntaxOff
5498   }
5499 }
```

The extra nodes

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

```

5500 \cs_generate_variant:Nn \dim_min:nn { v n }
5501 \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).

```

5502 \cs_new_protected:Npn \@@_create_extra_nodes:
5503 {
5504   \bool_if:nTF \l_@@_medium_nodes_bool
5505   {
5506     \bool_if:nTF \l_@@_large_nodes_bool
5507     \@@_create_medium_and_large_nodes:
5508     \@@_create_medium_nodes:
5509   }
5510   { \bool_if:NT \l_@@_large_nodes_bool \@@_create_large_nodes: }
5511 }
```

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`.

```

5512 \cs_new_protected:Npn \@@_computations_for_medium_nodes:
5513 {
5514   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
```

```

5515 {
5516   \dim_zero_new:c { l_@@_row\_@@_i: _min_dim }
5517   \dim_set_eq:cN { l_@@_row\_@@_i: _min_dim } \c_max_dim
5518   \dim_zero_new:c { l_@@_row\_@@_i: _max_dim }
5519   \dim_set:cn { l_@@_row\_@@_i: _max_dim } { - \c_max_dim }
5520 }
5521 \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
5522 {
5523   \dim_zero_new:c { l_@@_column\_@@_j: _min_dim }
5524   \dim_set_eq:cN { l_@@_column\_@@_j: _min_dim } \c_max_dim
5525   \dim_zero_new:c { l_@@_column\_@@_j: _max_dim }
5526   \dim_set:cn { l_@@_column\_@@_j: _max_dim } { - \c_max_dim }
5527 }

```

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

```

5528 \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
5529 {
5530   \int_step_variable:nnNn
5531     \l_@@_first_col_int \g_@@_col_total_int \@@_j:

```

If the cell (i - j) is empty or an implicit cell (that is to say a cell after implicit ampersands &) we don't update the dimensions we want to compute.

```

5532 {
5533   \cs_if_exist:cT
5534     { pgf @ sh @ ns @ \@@_env: - \@@_i: - \@@_j: }

```

We retrieve the coordinates of the anchor south west of the (normal) node of the cell (i - j). They will be stored in \pgf@x and \pgf@y.

```

5535 {
5536   \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { south-west }
5537   \dim_set:cn { l_@@_row\_@@_i: _min_dim }
5538   { \dim_min:vn { l_@@_row _ \@@_i: _min_dim } \pgf@y }
5539   \seq_if_in:NxF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
5540   {
5541     \dim_set:cn { l_@@_column _ \@@_j: _min_dim }
5542     { \dim_min:vn { l_@@_column _ \@@_j: _min_dim } \pgf@x }
5543   }

```

We retrieve the coordinates of the anchor north east of the (normal) node of the cell (i - j). They will be stored in \pgf@x and \pgf@y.

```

5544   \pgfpointanchor { \@@_env: - \@@_i: - \@@_j: } { north-east }
5545   \dim_set:cn { l_@@_row _ \@@_i: _ max_dim }
5546   { \dim_max:vn { l_@@_row _ \@@_i: _ max_dim } \pgf@y }
5547   \seq_if_in:NxF \g_@@_multicolumn_cells_seq { \@@_i: - \@@_j: }
5548   {
5549     \dim_set:cn { l_@@_column _ \@@_j: _ max_dim }
5550     { \dim_max:vn { l_@@_column _ \@@_j: _ max_dim } \pgf@x }
5551   }
5552 }
5553 }
5554 }

```

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

```

5555 \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
5556 {
5557   \dim_compare:nnNt
5558     { \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } } = \c_max_dim
5559   {
5560     \@@_qpoint:n { row - \@@_i: - base }
5561     \dim_set:cn { l_@@_row _ \@@_i: _ max _ dim } \pgf@y
5562     \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim } \pgf@y
5563   }
5564 }
5565 \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:

```

```

5566     {
5567         \dim_compare:nNnT
5568         { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } } = \c_max_dim
5569         {
5570             \@@_qpoint:n { col - \@@_j: }
5571             \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim } \pgf@y
5572             \dim_set:cn { l_@@_column _ \@@_j: _ min _ dim } \pgf@y
5573         }
5574     }
5575 }

```

Here is the command `\@@_create_medium_nodes:`. When this command is used, the “medium nodes” are created.

```

5576 \cs_new_protected:Npn \@@_create_medium_nodes:
5577 {
5578     \pgfpicture
5579     \pgfrememberpicturepositiononpagetrue
5580     \pgf@relevantforpicturesizefalse
5581     \@@_computations_for_medium_nodes:

```

Now, we can create the “medium nodes”. We use a command `\@@_create_nodes:` because this command will also be used for the creation of the “large nodes”.

```

5582     \tl_set:Nn \l_@@_suffix_tl { -medium }
5583     \@@_create_nodes:
5584     \endpgfpicture
5585 }

```

The command `\@@_create_large_nodes:` must be used when we want to create only the “large nodes” and not the medium ones⁷⁴. 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:`.

```

5586 \cs_new_protected:Npn \@@_create_large_nodes:
5587 {
5588     \pgfpicture
5589     \pgfrememberpicturepositiononpagetrue
5590     \pgf@relevantforpicturesizefalse
5591     \@@_computations_for_medium_nodes:
5592     \@@_computations_for_large_nodes:
5593     \tl_set:Nn \l_@@_suffix_tl { - large }
5594     \@@_create_nodes:
5595     \endpgfpicture
5596 }
5597 \cs_new_protected:Npn \@@_create_medium_and_large_nodes:
5598 {
5599     \pgfpicture
5600     \pgfrememberpicturepositiononpagetrue
5601     \pgf@relevantforpicturesizefalse
5602     \@@_computations_for_medium_nodes:

```

Now, we can create the “medium nodes”. We use a command `\@@_create_nodes:` because this command will also be used for the creation of the “large nodes”.

```

5603     \tl_set:Nn \l_@@_suffix_tl { - medium }
5604     \@@_create_nodes:
5605     \@@_computations_for_large_nodes:
5606     \tl_set:Nn \l_@@_suffix_tl { - large }
5607     \@@_create_nodes:
5608     \endpgfpicture
5609 }

```

⁷⁴If we want to create both, we have to use `\@@_create_medium_and_large_nodes:`

For “large nodes”, the exterior rows and columns don’t interfere. That’s why the loop over the columns will start at 1 and stop at `\c@jCol` (and not `\g_@@_col_total_int`). Idem for the rows.

```

5610 \cs_new_protected:Npn \@@_computations_for_large_nodes:
5611 {
5612   \int_set:Nn \l_@@_first_row_int 1
5613   \int_set:Nn \l_@@_first_col_int 1

```

We have to change the values of all the dimensions `l_@@_row_i_min_dim`, `l_@@_row_i_max_dim`, `l_@@_column_j_min_dim` and `l_@@_column_j_max_dim`.

```

5614   \int_step_variable:nNn { \c@iRow - 1 } \@@_i:
5615   {
5616     \dim_set:cn { l_@@_row _ \@@_i: _ min _ dim }
5617     {
5618       (
5619         \dim_use:c { l_@@_row _ \@@_i: _ min _ dim } +
5620         \dim_use:c { l_@@_row _ \@@_succ:n \@@_i: _ max _ dim }
5621       )
5622       / 2
5623     }
5624     \dim_set_eq:cc { l_@@_row _ \@@_succ:n \@@_i: _ max _ dim }
5625     { l_@@_row _ \@@_i: _ min _ dim }
5626   }
5627   \int_step_variable:nNn { \c@jCol - 1 } \@@_j:
5628   {
5629     \dim_set:cn { l_@@_column _ \@@_j: _ max _ dim }
5630     {
5631       (
5632         \dim_use:c { l_@@_column _ \@@_j: _ max _ dim } +
5633         \dim_use:c
5634         { l_@@_column _ \@@_succ:n \@@_j: _ min _ dim }
5635       )
5636       / 2
5637     }
5638     \dim_set_eq:cc { l_@@_column _ \@@_succ:n \@@_j: _ min _ dim }
5639     { l_@@_column _ \@@_j: _ max _ dim }
5640   }

```

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

```

5641   \dim_sub:cn
5642   { l_@@_column _ 1 _ min _ dim }
5643   \l_@@_left_margin_dim
5644   \dim_add:cn
5645   { l_@@_column _ \int_use:N \c@jCol _ max _ dim }
5646   \l_@@_right_margin_dim
5647 }

```

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).

```

5648 \cs_new_protected:Npn \@@_create_nodes:
5649 {
5650   \int_step_variable:nnNn \l_@@_first_row_int \g_@@_row_total_int \@@_i:
5651   {
5652     \int_step_variable:nnNn \l_@@_first_col_int \g_@@_col_total_int \@@_j:
5653     {

```

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

```

5654       \@@_pgf_rect_node:nnnnn
5655       { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
5656       { \dim_use:c { l_@@_column _ \@@_j: _ min _ dim } }

```

```

5657         { \dim_use:c { l_@@_row_ \@@_i: _min_dim } }
5658         { \dim_use:c { l_@@_column_ \@@_j: _max_dim } }
5659         { \dim_use:c { l_@@_row_ \@@_i: _max_dim } }
5660     \str_if_empty:NF \l_@@_name_str
5661     {
5662         \pgfnodealias
5663         { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
5664         { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
5665     }
5666 }
5667 }

```

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 .

```

5668     \seq_mapthread_function:NNN
5669     \g_@@_multicolumn_cells_seq
5670     \g_@@_multicolumn_sizes_seq
5671     \@@_node_for_multicolumn:nn
5672 }

5673 \cs_new_protected:Npn \@@_extract_coords_values: #1 - #2 \q_stop
5674 {
5675     \cs_set_nopar:Npn \@@_i: { #1 }
5676     \cs_set_nopar:Npn \@@_j: { #2 }
5677 }

```

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”).

```

5678 \cs_new_protected:Npn \@@_node_for_multicolumn:nn #1 #2
5679 {
5680     \@@_extract_coords_values: #1 \q_stop
5681     \@@_pgf_rect_node:nnnnn
5682     { \@@_env: - \@@_i: - \@@_j: \l_@@_suffix_tl }
5683     { \dim_use:c { l_@@_column_ \@@_j: _min_dim } }
5684     { \dim_use:c { l_@@_row_ \@@_i: _min_dim } }
5685     { \dim_use:c { l_@@_column_ \int_eval:n { \@@_j: +#2-1 } _max_dim } }
5686     { \dim_use:c { l_@@_row_ \@@_i: _max_dim } }
5687     \str_if_empty:NF \l_@@_name_str
5688     {
5689         \pgfnodealias
5690         { \l_@@_name_str - \@@_i: - \@@_j: \l_@@_suffix_tl }
5691         { \int_use:N \g_@@_env_int - \@@_i: - \@@_j: \l_@@_suffix_tl }
5692     }
5693 }

```

The blocks

The code deals with the command `\Block`. This command has no direct link with the environment `{NiceMatrixBlock}`.

The options of the command `\Block` will be analyzed first in the cell of the array (and once again when the block will be put in the array). Here is the set of keys for the first pass.

```

5694 \keys_define:nn { NiceMatrix / Block / FirstPass }
5695 {
5696     l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
5697     l .value_forbidden:n = true ,
5698     r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
5699     r .value_forbidden:n = true ,

```

```

5700   c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
5701   c .value_forbidden:n = true ,
5702   L .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
5703   L .value_forbidden:n = true ,
5704   R .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
5705   R .value_forbidden:n = true ,
5706   C .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
5707   C .value_forbidden:n = true ,
5708   t .code:n = \str_set:Nn \l_@@_vpos_of_block_tl t ,
5709   t .value_forbidden:n = true ,
5710   b .code:n = \str_set:Nn \l_@@_vpos_of_block_tl b ,
5711   b .value_forbidden:n = true ,
5712   color .tl_set:N = \l_@@_color_tl ,
5713   color .value_required:n = true ,
5714   respect-arraystretch .bool_set:N = \l_@@_respect_arraystretch_bool ,
5715   respect-arraystretch .default:n = true ,
5716 }

```

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.

```

5717 \NewExpandableDocumentCommand \@@_Block: { 0 { } m D < > { } +m }
5718 {

```

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).

```

5719   \peek_remove_spaces:n
5720   {
5721     \tl_if_blank:nTF { #2 }
5722     { \@@_Block_i 1-1 \q_stop }
5723     { \@@_Block_i #2 \q_stop }
5724     { #1 } { #3 } { #4 }
5725   }
5726 }

```

With the following construction, we extract the values of i and j in the first mandatory argument of the command.

```

5727 \cs_new:Npn \@@_Block_i #1-#2 \q_stop { \@@_Block_ii:nnnnn { #1 } { #2 } }

```

Now, the arguments have been extracted: `#1` is i (the number of rows of the block), `#2` is j (the number of columns of the block), `#3` is the list of `key=values` pairs, `#4` are the tokens to put before the math mode and the beginning of the small array of the block and `#5` is the label of the block.

```

5728 \cs_new_protected:Npn \@@_Block_ii:nnnnn #1 #2 #3 #4 #5
5729 {

```

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).

```

5730   \bool_lazy_or:nnTF
5731   { \tl_if_blank_p:n { #1 } }
5732   { \str_if_eq_p:nn { #1 } { * } }
5733   { \int_set:Nn \l_tmpa_int { 100 } }
5734   { \int_set:Nn \l_tmpa_int { #1 } }
5735   \bool_lazy_or:nnTF
5736   { \tl_if_blank_p:n { #2 } }
5737   { \str_if_eq_p:nn { #2 } { * } }
5738   { \int_set:Nn \l_tmpb_int { 100 } }
5739   { \int_set:Nn \l_tmpb_int { #2 } }

```

If the block is mono-column.

```

5740 \int_compare:nNnTF \l_tmpb_int = 1
5741 {
5742   \str_if_empty:NTF \l_@@_hpos_cell_str
5743     { \str_set:Nn \l_@@_hpos_block_str c }
5744     { \str_set_eq:NN \l_@@_hpos_block_str \l_@@_hpos_cell_str }
5745 }
5746 { \str_set:Nn \l_@@_hpos_block_str c }

```

The value of `\l_@@_hpos_block_str` may be modified by the keys of the command `\Block` that we will analyze now.

```

5747 \keys_set_known:nn { NiceMatrix / Block / FirstPass } { #3 }
5748 \tl_set:Nx \l_tmpa_tl
5749 {
5750   { \int_use:N \c@iRow }
5751   { \int_use:N \c@jCol }
5752   { \int_eval:n { \c@iRow + \l_tmpa_int - 1 } }
5753   { \int_eval:n { \c@jCol + \l_tmpb_int - 1 } }
5754 }

```

Now, `\l_tmpa_tl` contains an “object” corresponding to the position of the block with four components, each of them surrounded by curly brackets:

`{imin}{jmin}{imax}{jmax}`.

If the block is mono-column or mono-row, we have a special treatment. That’s why we have two macros: `\@@_Block_iv:nnnnn` and `\@@_Block_v:nnnnn` (the five arguments of those macros are provided by curryfication).

```

5755 \bool_if:nTF
5756 {
5757   (
5758     \int_compare_p:nNn { \l_tmpa_int } = 1
5759     ||
5760     \int_compare_p:nNn { \l_tmpb_int } = 1
5761   )
5762   && ! \tl_if_empty_p:n { #5 }

```

For the blocks mono-column, we will compose right now in a box in order to compute its width and take that width into account for the width of the column. However, if the column is a **X** column, we should not do that since the width is determined by another way. This should be the same for the **p**, **m** and **b** columns and we should modify that point. However, for the **X** column, it’s imperative. Otherwise, the process for the determination of the widths of the columns will be wrong.

```

5763   && ! \l_@@_X_column_bool
5764 }
5765 { \exp_args:Nxx \@@_Block_iv:nnnnn }
5766 { \exp_args:Nxx \@@_Block_v:nnnnn }
5767 { \l_tmpa_int } { \l_tmpb_int } { #3 } { #4 } { #5 }
5768 }

```

The following macro is for the case of a `\Block` which is mono-row or mono-column (or both). In that case, the content of the block is composed right now in a box (because we have to take into account the dimensions of that box for the width of the current column or the height and the depth of the current row). However, that box will be put in the array *after the construction of the array* (by using PGF).

```

5769 \cs_new_protected:Npn \@@_Block_iv:nnnnn #1 #2 #3 #4 #5
5770 {
5771   \int_gincr:N \g_@@_block_box_int
5772   \cs_set_protected_nopar:Npn \diagbox ##1 ##2
5773   {
5774     \tl_gput_right:Nx \g_@@_internal_code_after_tl
5775     {
5776       \@@_actually_diagbox:nnnnnn
5777       { \int_use:N \c@iRow }

```



```

5778         { \int_use:N \c@jCol }
5779         { \int_eval:n { \c@iRow + #1 - 1 } }
5780         { \int_eval:n { \c@jCol + #2 - 1 } }
5781         { \exp_not:n { ##1 } } } { \exp_not:n { ##2 } }
5782     }
5783 }
5784 \box_gclear_new:c
5785 { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
5786 \hbox_gset:cn
5787 { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
5788 {

```

For a mono-column block, if the user has specified a color for the column in the preamble of the array, we want to fix that color in the box we construct. We do that with `\set@color` and not `\color_ensure_current:` (in order to use `\color_ensure_current:` safely, you should load `l3backend` before the `\documentclass` with `\RequirePackage{expl3}`).

```

5789     \tl_if_empty:NTF \l_@@_color_tl
5790     { \int_compare:nNnT { #2 } = 1 \set@color }
5791     { \color { \l_@@_color_tl } }

```

If the block is mono-row, we use `\g_@@_row_style_tl` even if it has yet been used in the beginning of the cell where the command `\Block` has been issued because we want to be able to take into account a potential instruction of color of the font in `\g_@@_row_style_tl`.

```

5792     \int_compare:nNnT { #1 } = 1 \g_@@_row_style_tl
5793     \group_begin:
5794     \bool_if:NF \l_@@_respect_arraystretch_bool
5795     { \cs_set:Npn \arraystretch { 1 } }
5796     \dim_zero:N \extrarowheight
5797     #4

```

If the box is rotated (the key `\rotate` may be in the previous `#4`), the tabular used for the content of the cell will be constructed with a format `c`. In the other cases, the tabular will be constructed with a format equal to the key of position of the box. In other words: the alignment internal to the tabular is the same as the external alignment of the tabular (that is to say the position of the block in its zone of merged cells).

```

5798     \bool_if:NT \g_@@_rotate_bool { \str_set:Nn \l_@@_hpos_block_str c }
5799     \bool_if:NTF \l_@@_NiceTabular_bool
5800     {
5801         \bool_lazy_all:nTF
5802         {
5803             { \int_compare_p:nNn { #2 } = 1 }
5804             { \dim_compare_p:n { \l_@@_col_width_dim >= \c_zero_dim } }
5805             { ! \l_@@_respect_arraystretch_bool }
5806         }

```

When the block is mono-column in a column with a fixed width (eg `p{3cm}`).

```

5807     {
5808         \begin { minipage } [ \l_@@_vpos_of_block_tl ]
5809         { \l_@@_col_width_dim }
5810         \str_case:Nn \l_@@_hpos_block_str
5811         {
5812             c \centering
5813             r \raggedleft
5814             l \raggedright
5815         }
5816         #5
5817         \end { minipage }
5818     }
5819     {
5820         \use:x
5821         {
5822             \exp_not:N \begin { tabular } [ \l_@@_vpos_of_block_tl ]
5823             { @ { } \l_@@_hpos_block_str @ { } }
5824         }

```

```

5825         #5
5826     \end { tabular }
5827 }
5828 }
5829 {
5830     \c_math_toggle_token
5831     \use:x
5832     {
5833         \exp_not:N \begin { array } [ \l_@@_vpos_of_block_tl ]
5834         { @ { } \l_@@_hpos_block_str @ { } }
5835     }
5836     #5
5837     \end { array }
5838     \c_math_toggle_token
5839 }
5840 \group_end:
5841 }
5842 \bool_if:NT \g_@@_rotate_bool
5843 {
5844     \box_grotate:cn
5845     { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
5846     { 90 }
5847     \bool_gset_false:N \g_@@_rotate_bool
5848 }

```

If we are in a mono-column block, we take into account the width of that block for the width of the column.

```

5849 \int_compare:nNnT { #2 } = 1
5850 {
5851     \dim_gset:Nn \g_@@_blocks_wd_dim
5852     {
5853         \dim_max:nn
5854         \g_@@_blocks_wd_dim
5855         {
5856             \box_wd:c
5857             { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
5858         }
5859     }
5860 }

```

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.

```

5861 \int_compare:nNnT { #1 } = 1
5862 {
5863     \dim_gset:Nn \g_@@_blocks_ht_dim
5864     {
5865         \dim_max:nn
5866         \g_@@_blocks_ht_dim
5867         {
5868             \box_ht:c
5869             { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
5870         }
5871     }
5872     \dim_gset:Nn \g_@@_blocks_dp_dim
5873     {
5874         \dim_max:nn
5875         \g_@@_blocks_dp_dim
5876         {
5877             \box_dp:c
5878             { g_@@_block _ box _ \int_use:N \g_@@_block_box_int _ box }
5879         }
5880     }
5881 }
5882 \seq_gput_right:Nx \g_@@_blocks_seq

```

```

5883 {
5884   \l_tmpa_tl

```

In the list of options #3, maybe there is a key for the horizontal alignment (l, r or c). In that case, that key has been read and stored in \l_@@_hpos_block_str. However, maybe there were no key of the horizontal alignment and that's why we put a key corresponding to the value of \l_@@_hpos_block_str, which is fixed by the type of current column.

```

5885   { \exp_not:n { #3 } , \l_@@_hpos_block_str }
5886   {
5887     \box_use_drop:c
5888     { g_@@_ block _ box _ \int_use:N \g_@@_block_box_int _ box }
5889   }
5890 }
5891 }

```

The following macro is for the standard case, where the block is not mono-row and not mono-column. In that case, the content of the block is *not* composed right now in a box. The composition in a box will be done further, just after the construction of the array.

```

5892 \cs_new_protected:Npn \@@_Block_v:nnnnn #1 #2 #3 #4 #5
5893 {
5894   \seq_gput_right:Nx \g_@@_blocks_seq
5895   {
5896     \l_tmpa_tl
5897     { \exp_not:n { #3 } }
5898     \exp_not:n
5899     {
5900       {
5901         \bool_if:NTF \l_@@_NiceTabular_bool
5902         {
5903           \group_begin:
5904           \bool_if:NF \l_@@_respect_arraystretch_bool
5905             { \cs_set:Npn \arraystretch { 1 } }
5906           \dim_zero:N \extrarowheight
5907           #4

```

If the box is rotated (the key \rotate may be in the previous #4), the tabular used for the content of the cell will be constructed with a format c. In the other cases, the tabular will be constructed with a format equal to the key of position of the box. In other words: the alignment internal to the tabular is the same as the external alignment of the tabular (that is to say the position of the block in its zone of merged cells).

```

5908     \bool_if:NT \g_@@_rotate_bool
5909     { \str_set:Nn \l_@@_hpos_block_str c }
5910     \use:x
5911     {
5912       \exp_not:N \begin { tabular } [ \l_@@_vpos_of_block_tl ]
5913       { @ { } \l_@@_hpos_block_str @ { } }
5914     }
5915     #5
5916   \end { tabular }
5917   \group_end:
5918 }
5919 {
5920   \group_begin:
5921   \bool_if:NF \l_@@_respect_arraystretch_bool
5922     { \cs_set:Npn \arraystretch { 1 } }
5923   \dim_zero:N \extrarowheight
5924   #4
5925   \bool_if:NT \g_@@_rotate_bool
5926   { \str_set:Nn \l_@@_hpos_block_str c }
5927   \c_math_toggle_token
5928   \use:x
5929   {
5930     \exp_not:N \begin { array } [ \l_@@_vpos_of_block_tl ]

```

```

5931         { @ { } \l_@@_hpos_block_str @ { } }
5932     }
5933     #5
5934     \end { array }
5935     \c_math_toggle_token
5936     \group_end:
5937 }
5938 }
5939 }
5940 }
5941 }

```

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).

```

5942 \keys_define:nn { NiceMatrix / Block / SecondPass }
5943 {
5944     tikz .code:n =
5945         \bool_if:NTF \c_@@_tikz_loaded_bool
5946         { \seq_put_right:Nn \l_@@_tikz_seq { { #1 } } }
5947         { \@@_error:n { tikz-key-without~tikz } } ,
5948     tikz .value_required:n = true ,
5949     fill .tl_set:N = \l_@@_fill_tl ,
5950     fill .value_required:n = true ,
5951     draw .tl_set:N = \l_@@_draw_tl ,
5952     draw .default:n = default ,
5953     rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
5954     rounded-corners .default:n = 4 pt ,
5955     color .code:n = \color { #1 } \tl_set:Nn \l_@@_draw_tl { #1 } ,
5956     color .value_required:n = true ,
5957     borders .clist_set:N = \l_@@_borders_clist ,
5958     borders .value_required:n = true ,
5959     hvlines .meta:n = { vlines = #1 , hlines = #1 } ,
5960     vlines .bool_set:N = \l_@@_vlines_block_bool ,
5961     vlines .default:n = true ,
5962     hlines .bool_set:N = \l_@@_hlines_block_bool ,
5963     hlines .default:n = true ,
5964     line-width .dim_set:N = \l_@@_line_width_dim ,
5965     line-width .value_required:n = true ,
5966     l .code:n = \str_set:Nn \l_@@_hpos_block_str l ,
5967     l .value_forbidden:n = true ,
5968     r .code:n = \str_set:Nn \l_@@_hpos_block_str r ,
5969     r .value_forbidden:n = true ,
5970     c .code:n = \str_set:Nn \l_@@_hpos_block_str c ,
5971     c .value_forbidden:n = true ,
5972     L .code:n = \str_set:Nn \l_@@_hpos_block_str l
5973         \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
5974     L .value_forbidden:n = true ,
5975     R .code:n = \str_set:Nn \l_@@_hpos_block_str r
5976         \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
5977     R .value_forbidden:n = true ,
5978     C .code:n = \str_set:Nn \l_@@_hpos_block_str c
5979         \bool_set_true:N \l_@@_hpos_of_block_cap_bool ,
5980     C .value_forbidden:n = true ,
5981     t .code:n = \str_set:Nn \l_@@_vpos_of_block_tl t ,
5982     t .value_forbidden:n = true ,
5983     b .code:n = \str_set:Nn \l_@@_vpos_of_block_tl b ,
5984     b .value_forbidden:n = true ,
5985     name .tl_set:N = \l_@@_block_name_str ,
5986     name .value_required:n = true ,
5987     name .initial:n = ,
5988     respect-arraystretch .bool_set:N = \l_@@_respect_arraystretch_bool ,

```

```

5989   respect-arraystretch .default:n = true ,
5990   unknown .code:n = \@@_error:n { Unknown-key-for-Block }
5991 }

```

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.

```

5992 \cs_new_protected:Npn \@@_draw_blocks:
5993 {
5994   \cs_set_eq:NN \ialign \@@_old_ialign:
5995   \seq_map_inline:Nn \g_@@_blocks_seq { \@@_Block_iv:nnnnnn #1 }
5996 }
5997 \cs_new_protected:Npn \@@_Block_iv:nnnnnn #1 #2 #3 #4 #5 #6
5998 {

```

The integer `\l_@@_last_row_int` will be the last row of the block and `\l_@@_last_col_int` its last column.

```

5999   \int_zero_new:N \l_@@_last_row_int
6000   \int_zero_new:N \l_@@_last_col_int

```

We remind that the first mandatory argument of the command `\Block` is the size of the block with the special format $i-j$. However, the user is allowed to omit i or j (or both). This will be interpreted as: the last row (resp. column) of the block will be the last row (resp. column) of the block (without the potential exterior row—resp. column—of the array). By convention, this is stored in `\g_@@_blocks_seq` as a number of rows (resp. columns) for the block equal to 100. That's what we detect now.

```

6001   \int_compare:nNnTF { #3 } > { 99 }
6002   { \int_set_eq:NN \l_@@_last_row_int \c{iRow }
6003     { \int_set:Nn \l_@@_last_row_int { #3 } }
6004   \int_compare:nNnTF { #4 } > { 99 }
6005   { \int_set_eq:NN \l_@@_last_col_int \c{jCol }
6006     { \int_set:Nn \l_@@_last_col_int { #4 } }
6007   \int_compare:nNnTF \l_@@_last_col_int > \g_@@_col_total_int
6008   {
6009     \int_compare:nTF
6010     { \l_@@_last_col_int <= \g_@@_static_num_of_col_int }
6011     {
6012       \msg_error:nnnn { nicematrix } { Block-too-large~2 } { #1 } { #2 }
6013       \@@_msg_redirect_name:nn { Block-too-large~2 } { none }
6014       \group_begin:
6015       \globaldefs = 1
6016       \@@_msg_redirect_name:nn { columns-not-used } { none }
6017       \group_end:
6018     }
6019     { \msg_error:nnnn { nicematrix } { Block-too-large~1 } { #1 } { #2 } }
6020   }
6021   {
6022     \int_compare:nNnTF \l_@@_last_row_int > \g_@@_row_total_int
6023     { \msg_error:nnnn { nicematrix } { Block-too-large~1 } { #1 } { #2 } }
6024     { \@@_Block_v:nnnnnn { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } }
6025   }
6026 }
6027 \cs_new_protected:Npn \@@_Block_v:nnnnnn #1 #2 #3 #4 #5 #6
6028 {

```

The group is for the keys.

```

6029   \group_begin:
6030   \keys_set:nn { NiceMatrix / Block / SecondPass } { #5 }
6031   \bool_if:NT \l_@@_vlines_block_bool
6032   {
6033     \tl_gput_right:Nx \g_nicematrix_code_after_tl
6034     {

```

```

6035         \l_@@_vlines_block:nnn
6036         { \exp_not:n { #5 } }
6037         { #1 - #2 }
6038         { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
6039     }
6040 }
6041 \bool_if:NT \l_@@_hlines_block_bool
6042 {
6043     \tl_gput_right:Nx \g_nicematrix_code_after_tl
6044     {
6045         \l_@@_hlines_block:nnn
6046         { \exp_not:n { #5 } }
6047         { #1 - #2 }
6048         { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
6049     }
6050 }
6051 \bool_if:nT
6052 { ! \l_@@_vlines_block_bool && ! \l_@@_hlines_block_bool }
6053 {

```

The sequence of the positions of the blocks (excepted the blocks with the key `hvlines`) will be used when drawing the rules (in fact, there is also the `\multicolumn` and the `\diagbox` in that sequence).

```

6054     \seq_gput_left:Nx \g_@@_pos_of_blocks_seq
6055     { { #1 } { #2 } { #3 } { #4 } { \l_@@_block_name_str } }
6056 }
6057 \tl_if_empty:NF \l_@@_draw_tl
6058 {
6059     \tl_gput_right:Nx \g_nicematrix_code_after_tl
6060     {
6061         \l_@@_stroke_block:nnn
6062         { \exp_not:n { #5 } }
6063         { #1 - #2 }
6064         { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
6065     }
6066     \seq_gput_right:Nn \g_@@_pos_of_stroken_blocks_seq
6067     { { #1 } { #2 } { #3 } { #4 } }
6068 }
6069 \clist_if_empty:NF \l_@@_borders_clist
6070 {
6071     \tl_gput_right:Nx \g_nicematrix_code_after_tl
6072     {
6073         \l_@@_stroke_borders_block:nnn
6074         { \exp_not:n { #5 } }
6075         { #1 - #2 }
6076         { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
6077     }
6078 }
6079 \tl_if_empty:NF \l_@@_fill_tl
6080 {

```

The command `\l_@@_extract_brackets` will extract the potential specification of color space at the beginning of `\l_@@_fill_tl` and store it in `\l_tmpa_tl` and store the color itself in `\l_tmpb_tl`.

```

6081     \exp_last_unbraced:NV \l_@@_extract_brackets \l_@@_fill_tl \q_stop
6082     \tl_gput_right:Nx \g_nicematrix_code_before_tl
6083     {
6084         \exp_not:N \roundedrectanglecolor
6085         [ \l_tmpa_tl ]
6086         { \exp_not:V \l_tmpb_tl }
6087         { #1 - #2 }
6088         { \int_use:N \l_@@_last_row_int - \int_use:N \l_@@_last_col_int }
6089         { \dim_use:N \l_@@_rounded_corners_dim }
6090     }
6091 }

```

```

6092 \seq_if_empty:NF \l_@@_tikz_seq
6093 {
6094   \tl_gput_right:Nx \g_nicematrix_code_before_tl
6095   {
6096     \@@_block_tikz:nnnnn
6097     { #1 }
6098     { #2 }
6099     { \int_use:N \l_@@_last_row_int }
6100     { \int_use:N \l_@@_last_col_int }
6101     { \seq_use:Nn \l_@@_tikz_seq { , } }
6102   }
6103 }

6104 \cs_set_protected_nopar:Npn \diagbox ##1 ##2
6105 {
6106   \tl_gput_right:Nx \g_@@_internal_code_after_tl
6107   {
6108     \@@_actually_diagbox:nnnnnn
6109     { #1 }
6110     { #2 }
6111     { \int_use:N \l_@@_last_row_int }
6112     { \int_use:N \l_@@_last_col_int }
6113     { \exp_not:n { ##1 } } { \exp_not:n { ##2 } }
6114   }
6115 }

6116 \hbox_set:Nn \l_@@_cell_box { \set@color #6 }
6117 \bool_if:NT \g_@@_rotate_bool \@@_rotate_cell_box:

```

Let's consider the following `{NiceTabular}`. Because of the instruction `!\hspace{1cm}` in the preamble which increases the space between the columns (by adding, in fact, that space to the previous column, that is to say the second column of the tabular), we will create *two* nodes relative to the block: the node `1-1-block` and the node `1-1-block-short`.

```

\begin{NiceTabular}{cc!\hspace{1cm}}c}
\Block{2-2}{our block} &      & one & \\
                        &      & two & \\
three                  & four & five & \\
six                    & seven & eight & \\
\end{NiceTabular}

```

We highlight the node `1-1-block`

our block		one
		two
three	four	five
six	seven	eight

We highlight the node `1-1-block-short`

our block		one
		two
three	four	five
six	seven	eight

The construction of the node corresponding to the merged cells.

```

6118 \pgfpicture
6119 \pgfrememberpicturepositiononpagetrue
6120 \pgf@relevantforpicturesizefalse
6121 \@@_qpoint:n { row - #1 }
6122 \dim_set_eq:NN \l_tmpa_dim \pgf@y
6123 \@@_qpoint:n { col - #2 }
6124 \dim_set_eq:NN \l_tmpb_dim \pgf@x
6125 \@@_qpoint:n { row - \@@_succ:n { \l_@@_last_row_int } }
6126 \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
6127 \@@_qpoint:n { col - \@@_succ:n { \l_@@_last_col_int } }
6128 \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x

```

We construct the node for the block with the name (#1-#2-block).

The function `\@@pgf_rect_node:nnnnn` takes in as arguments the name of the node and the four coordinates of two opposite corner points of the rectangle.

```

6129 \@@pgf_rect_node:nnnnn
6130 { \@@_env: - #1 - #2 - block }
6131 \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
6132 \str_if_empty:NF \l_@@_block_name_str
6133 {
6134   \pgfnodealias
6135   { \@@_env: - \l_@@_block_name_str }
6136   { \@@_env: - #1 - #2 - block }
6137   \str_if_empty:NF \l_@@_name_str
6138   {
6139     \pgfnodealias
6140     { \l_@@_name_str - \l_@@_block_name_str }
6141     { \@@_env: - #1 - #2 - block }
6142   }
6143 }

```

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.

```

6144 \bool_if:NF \l_@@_hpos_of_block_cap_bool
6145 {
6146   \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.

```

6147 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
6148 {

```

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.

```

6149 \cs_if_exist:cT
6150 { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
6151 {
6152   \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
6153   {
6154     \pgfpointanchor { \@@_env: - ##1 - #2 } { west }
6155     \dim_set:Nn \l_tmpb_dim { \dim_min:nn \l_tmpb_dim \pgf@x }
6156   }
6157 }
6158 }

```

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.

```

6159 \dim_compare:nNnT \l_tmpb_dim = \c_max_dim
6160 {
6161   \@@_qpoint:n { col - #2 }
6162   \dim_set_eq:NN \l_tmpb_dim \pgf@x
6163 }
6164 \dim_set:Nn \l_@@_tmpd_dim { - \c_max_dim }
6165 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
6166 {
6167   \cs_if_exist:cT
6168   { pgf @ sh @ ns @ \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
6169   {
6170     \seq_if_in:NnF \g_@@_multicolumn_cells_seq { ##1 - #2 }
6171     {
6172       \pgfpointanchor
6173       { \@@_env: - ##1 - \int_use:N \l_@@_last_col_int }
6174       { east }

```



```

6175         \dim_set:Nn \l_@@_tmpd_dim { \dim_max:nn \l_@@_tmpd_dim \pgf@x }
6176     }
6177 }
6178 }
6179 \dim_compare:nNnT \l_@@_tmpd_dim = { - \c_max_dim }
6180 {
6181     \@@_qpoint:n { col - \@@_succ:n { \l_@@_last_col_int } }
6182     \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
6183 }
6184 \@@_pgf_rect_node:nnnnn
6185 { \@@_env: - #1 - #2 - block - short }
6186 \l_tmpb_dim \l_tmpa_dim \l_@@_tmpd_dim \l_@@_tmpc_dim
6187 }

```

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.

```

6188 \bool_if:NT \l_@@_medium_nodes_bool
6189 {
6190     \@@_pgf_rect_node:nnn
6191     { \@@_env: - #1 - #2 - block - medium }
6192     { \pgfpointanchor { \@@_env: - #1 - #2 - medium } { north-west } }
6193     {
6194         \pgfpointanchor
6195         { \@@_env:
6196             - \int_use:N \l_@@_last_row_int
6197             - \int_use:N \l_@@_last_col_int - medium
6198         }
6199         { south-east }
6200     }
6201 }

```

Now, we will put the label of the block beginning with the case of a `\Block` of one row.

```

6202 \int_compare:nNnTF { #1 } = { #3 }
6203 {

```

We take into account the case of a block of one row in the “first row” or the “last row”.

```

6204     \int_compare:nNnTF { #1 } = 0
6205     { \l_@@_code_for_first_row_tl }
6206     {
6207         \int_compare:nNnT { #1 } = \l_@@_last_row_int
6208         \l_@@_code_for_last_row_tl
6209     }

```

If the block has only one row, we want the label of the block perfectly aligned on the baseline of the row. That’s why we have constructed a `\pgfcoordinate` on the baseline of the row, in the first column of the array. Now, we retrieve the y -value of that node and we store it in `\l_tmpa_dim`.

```

6210     \pgfextracty \l_tmpa_dim { \@@_qpoint:n { row - #1 - base } }

```

We retrieve (in `\pgf@x`) the x -value of the center of the block.

```

6211 \pgfpointanchor
6212 {
6213     \@@_env: - #1 - #2 - block
6214     \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
6215 }
6216 {
6217     \str_case:Vn \l_@@_hpos_block_str
6218     {
6219         c { center }
6220         l { west }
6221         r { east }
6222     }
6223 }

```

We put the label of the block which has been composed in `\l_@@_cell_box`.

```

6224 \pgftransformshift { \pgfpoint \pgf@x \l_tmpa_dim }
6225 \pgfset { inner-sep = \c_zero_dim }
6226 \pgfnode
6227 { rectangle }
6228 {
6229 \str_case:Vn \l_@@_hpos_block_str
6230 {
6231 c { base }
6232 l { base-west }
6233 r { base-east }
6234 }
6235 }
6236 { \box_use_drop:N \l_@@_cell_box } { } { }
6237 }

```

If the number of rows is different of 1, we will put the label of the block by using the short node (the label of the block has been composed in `\l_@@_cell_box`).

```

6238 {

```

If we are in the first column, we must put the block as if it was with the key `r`.

```

6239 \int_compare:nNnT { #2 } = 0
6240 { \str_set:Nn \l_@@_hpos_block_str r }
6241 \bool_if:nT \g_@@_last_col_found_bool
6242 {
6243 \int_compare:nNnT { #2 } = \g_@@_col_total_int
6244 { \str_set:Nn \l_@@_hpos_block_str l }
6245 }
6246 \pgftransformshift
6247 {
6248 \pgfpointanchor
6249 {
6250 \@@_env: - #1 - #2 - block
6251 \bool_if:NF \l_@@_hpos_of_block_cap_bool { - short }
6252 }
6253 {
6254 \str_case:Vn \l_@@_hpos_block_str
6255 {
6256 c { center }
6257 l { west }
6258 r { east }
6259 }
6260 }
6261 }
6262 \pgfset { inner-sep = \c_zero_dim }
6263 \pgfnode
6264 { rectangle }
6265 {
6266 \str_case:Vn \l_@@_hpos_block_str
6267 {
6268 c { center }
6269 l { west }
6270 r { east }
6271 }
6272 }
6273 { \box_use_drop:N \l_@@_cell_box } { } { }
6274 }
6275 \endpgfpicture
6276 \group_end:
6277 }

```

```

6278 \NewDocumentCommand \@@_extract_brackets { 0 { } }
6279 {

```

```

6280 \tl_set:Nn \l_tmpa_tl { #1 }
6281 \@@_store_in_tmptb_tl
6282 }
6283 \cs_new_protected:Npn \@@_store_in_tmptb_tl #1 \q_stop
6284 { \tl_set:Nn \l_tmptb_tl { #1 } }

```

The first argument of `\@@_stroke_block:nnn` is a list of options for the rectangle that you will stroke. The second argument is the upper-left cell of the block (with, as usual, the syntax $i-j$) and the third is the last cell of the block (with the same syntax).

```

6285 \cs_new_protected:Npn \@@_stroke_block:nnn #1 #2 #3
6286 {
6287   \group_begin:
6288   \tl_clear:N \l_@@_draw_tl
6289   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
6290   \keys_set_known:nn { NiceMatrix / BlockStroke } { #1 }
6291   \pgfpicture
6292   \pgfrememberpicturepositiononpagetrue
6293   \pgf@relevantforpicturesizefalse
6294   \tl_if_empty:NF \l_@@_draw_tl
6295   {

```

If the user has used the key `color` of the command `\Block` without value, the color fixed by `\arrayrulecolor` is used.

```

6296   \str_if_eq:VnTF \l_@@_draw_tl { default }
6297   { \CT@arc@ }
6298   { \exp_args:NV \pgfsetstrokecolor \l_@@_draw_tl }
6299   }
6300   \pgfsetcornersarced
6301   {
6302     \pgfpoint
6303     { \dim_use:N \l_@@_rounded_corners_dim }
6304     { \dim_use:N \l_@@_rounded_corners_dim }
6305   }
6306   \@@_cut_on_hyphen:w #2 \q_stop
6307   \bool_lazy_and:nnT
6308   { \int_compare_p:n { \l_tmpa_tl <= \c@iRow } }
6309   { \int_compare_p:n { \l_tmptb_tl <= \c@jCol } }
6310   {
6311     \@@_qpoint:n { row - \l_tmpa_tl }
6312     \dim_set:Nn \l_tmptb_dim { \pgf@y }
6313     \@@_qpoint:n { col - \l_tmptb_tl }
6314     \dim_set:Nn \l_@@_tmpc_dim { \pgf@x }
6315     \@@_cut_on_hyphen:w #3 \q_stop
6316     \int_compare:nNnT \l_tmpa_tl > \c@iRow
6317     { \tl_set:Nx \l_tmpa_tl { \int_use:N \c@iRow } }
6318     \int_compare:nNnT \l_tmptb_tl > \c@jCol
6319     { \tl_set:Nx \l_tmptb_tl { \int_use:N \c@jCol } }
6320     \@@_qpoint:n { row - \@@_succ:n \l_tmpa_tl }
6321     \dim_set:Nn \l_tmpa_dim { \pgf@y }
6322     \@@_qpoint:n { col - \@@_succ:n \l_tmptb_tl }
6323     \dim_set:Nn \l_@@_tmpd_dim { \pgf@x }
6324     \pgfpathrectanglecorners
6325     { \pgfpoint \l_@@_tmpc_dim \l_tmptb_dim }
6326     { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
6327     \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }

```

We can't use `\pgfusepathqstroke` because of the key `rounded-corners`.

```

6328   \pgfusepath { stroke }
6329   }
6330   \endpgfpicture
6331   \group_end:
6332   }

```

Here is the set of keys for the command `\@@_stroke_block:nnn`.

```

6333 \keys_define:nn { NiceMatrix / BlockStroke }
6334 {
6335   color .tl_set:N = \l_@@_draw_tl ,
6336   draw .tl_set:N = \l_@@_draw_tl ,
6337   draw .default:n = default ,
6338   line-width .dim_set:N = \l_@@_line_width_dim ,
6339   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
6340   rounded-corners .default:n = 4 pt
6341 }

```

The first argument of `\@@_hvlines_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).

```

6342 \cs_new_protected:Npn \@@_vlines_block:nnn #1 #2 #3
6343 {
6344   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
6345   \keys_set_known:nn { NiceMatrix / BlockBorders } { #1 }
6346   \@@_cut_on_hyphen:w #2 \q_stop
6347   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
6348   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
6349   \@@_cut_on_hyphen:w #3 \q_stop
6350   \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
6351   \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
6352   \int_step_inline:nnn \l_@@_tmpd_tl \l_tmpb_tl
6353   {
6354     \use:x
6355     {
6356       \@@_vline:n
6357       {
6358         position = ##1 ,
6359         start = \l_@@_tmpc_tl ,
6360         end = \@@_pred:n \l_tmpa_tl
6361       }
6362     }
6363   }
6364 }
6365 \cs_new_protected:Npn \@@_hlines_block:nnn #1 #2 #3
6366 {
6367   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
6368   \keys_set_known:nn { NiceMatrix / BlockBorders } { #1 }
6369   \@@_cut_on_hyphen:w #2 \q_stop
6370   \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
6371   \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
6372   \@@_cut_on_hyphen:w #3 \q_stop
6373   \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
6374   \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
6375   \int_step_inline:nnn \l_@@_tmpc_tl \l_tmpa_tl
6376   {
6377     \use:x
6378     {
6379       \@@_hline:n
6380       {
6381         position = ##1 ,
6382         start = \l_@@_tmpd_tl ,
6383         end = \int_eval:n { \l_tmpb_tl - 1 }
6384       }
6385     }
6386   }
6387 }

```

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).

```

6388 \cs_new_protected:Npn \@@_stroke_borders_block:nnn #1 #2 #3
6389 {
6390   \dim_set_eq:NN \l_@@_line_width_dim \arrayrulewidth
6391   \keys_set_known:nn { NiceMatrix / BlockBorders } { #1 }
6392   \dim_compare:nNnTF \l_@@_rounded_corners_dim > \c_zero_dim
6393     { \@@_error:n { borders~forbidden } }
6394     {
6395       \clist_map_inline:Nn \l_@@_borders_clist
6396       {
6397         \clist_if_in:nnF { top , bottom , left , right } { ##1 }
6398         { \@@_error:nn { bad-border } { ##1 } }
6399       }
6400       \@@_cut_on_hyphen:w #2 \q_stop
6401       \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
6402       \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
6403       \@@_cut_on_hyphen:w #3 \q_stop
6404       \tl_set:Nx \l_tmpa_tl { \int_eval:n { \l_tmpa_tl + 1 } }
6405       \tl_set:Nx \l_tmpb_tl { \int_eval:n { \l_tmpb_tl + 1 } }
6406       \pgfpicture
6407       \pgfrememberpicturepositiononpagetrue
6408       \pgf@relevantforpicturesizefalse
6409       \CT@arc@
6410       \pgfsetlinewidth { 1.1 \l_@@_line_width_dim }
6411       \clist_if_in:NnT \l_@@_borders_clist { right }
6412       { \@@_stroke_vertical:n \l_tmpb_tl }
6413       \clist_if_in:NnT \l_@@_borders_clist { left }
6414       { \@@_stroke_vertical:n \l_@@_tmpd_tl }
6415       \clist_if_in:NnT \l_@@_borders_clist { bottom }
6416       { \@@_stroke_horizontal:n \l_tmpa_tl }
6417       \clist_if_in:NnT \l_@@_borders_clist { top }
6418       { \@@_stroke_horizontal:n \l_@@_tmpc_tl }
6419       \endpgfpicture
6420     }
6421 }

```

The following command is used to stroke the left border and the right border. The argument #1 is the number of column (in the sense of the `col` node).

```

6422 \cs_new_protected:Npn \@@_stroke_vertical:n #1
6423 {
6424   \@@_qpoint:n \l_@@_tmpc_tl
6425   \dim_set:Nn \l_tmpb_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
6426   \@@_qpoint:n \l_tmpa_tl
6427   \dim_set:Nn \l_@@_tmpc_dim { \pgf@y + 0.5 \l_@@_line_width_dim }
6428   \@@_qpoint:n { #1 }
6429   \pgfpathmoveto { \pgfpoint \pgf@x \l_tmpb_dim }
6430   \pgfpathlineto { \pgfpoint \pgf@x \l_@@_tmpc_dim }
6431   \pgfusepathqstroke
6432 }

```

The following command is used to stroke the top border and the bottom border. The argument #1 is the number of row (in the sense of the `row` node).

```

6433 \cs_new_protected:Npn \@@_stroke_horizontal:n #1
6434 {
6435   \@@_qpoint:n \l_@@_tmpd_tl
6436   \clist_if_in:NnTF \l_@@_borders_clist { left }
6437     { \dim_set:Nn \l_tmpa_dim { \pgf@x - 0.5 \l_@@_line_width_dim } }
6438     { \dim_set:Nn \l_tmpa_dim { \pgf@x + 0.5 \l_@@_line_width_dim } }
6439   \@@_qpoint:n \l_tmpb_tl
6440   \dim_set:Nn \l_tmpb_dim { \pgf@x + 0.5 \l_@@_line_width_dim }
6441   \@@_qpoint:n { #1 }
6442   \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }

```

```

6443 \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
6444 \pgfusepathqstroke
6445 }

```

Here is the set of keys for the command `\@@_stroke_borders_block:nnn`.

```

6446 \keys_define:nn { NiceMatrix / BlockBorders }
6447 {
6448   borders .clist_set:N = \l_@@_borders_clist ,
6449   rounded-corners .dim_set:N = \l_@@_rounded_corners_dim ,
6450   rounded-corners .default:n = 4 pt ,
6451   line-width .dim_set:N = \l_@@_line_width_dim
6452 }

```

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.

```

6453 \cs_new_protected:Npn \@@_block_tikz:nnnnn #1 #2 #3 #4 #5
6454 {
6455   \begin { tikzpicture }
6456   \clist_map_inline:nn { #5 }
6457   {
6458     \path [ ##1 ]
6459           ( #1 -| #2 ) rectangle ( \@@_succ:n { #3 } -| \@@_succ:n { #4 } ) ;
6460   }
6461   \end { tikzpicture }
6462 }

```

How to draw the dotted lines transparently

```

6463 \cs_set_protected:Npn \@@_renew_matrix:
6464 {
6465   \RenewDocumentEnvironment { pmatrix } { } {
6466     { \pNiceMatrix }
6467     { \endpNiceMatrix }
6468   \RenewDocumentEnvironment { vmatrix } { } {
6469     { \vNiceMatrix }
6470     { \endvNiceMatrix }
6471   \RenewDocumentEnvironment { Vmatrix } { } {
6472     { \VNiceMatrix }
6473     { \endVNiceMatrix }
6474   \RenewDocumentEnvironment { bmatrix } { } {
6475     { \bNiceMatrix }
6476     { \endbNiceMatrix }
6477   \RenewDocumentEnvironment { Bmatrix } { } {
6478     { \BNiceMatrix }
6479     { \endBNiceMatrix }
6480 }

```

Automatic arrays

```

6481 \cs_new_protected:Npn \@@_set_size:n #1-#2 \q_stop
6482 {
6483   \int_set:Nn \l_@@_nb_rows_int { #1 }
6484   \int_set:Nn \l_@@_nb_cols_int { #2 }
6485 }

```

We will extract the potential keys `l`, `r` and `c` and pass the other keys to the environment `{NiceArrayWithDelims}`.

```

6486 \keys_define:nn { NiceMatrix / Auto }
6487 {

```

```

6488   l .code:n = \tl_set:Nn \l_@@_type_of_col_tl l ,
6489   r .code:n = \tl_set:Nn \l_@@_type_of_col_tl r ,
6490   c .code:n = \tl_set:Nn \l_@@_type_of_col_tl c
6491 }
6492 \NewDocumentCommand \AutoNiceMatrixWithDelims { m m O { } m O { } m ! O { } }
6493 {
6494   \int_zero_new:N \l_@@_nb_rows_int
6495   \int_zero_new:N \l_@@_nb_cols_int
6496   \@@_set_size:n #4 \q_stop

```

The group is for the protection of `\l_@@_type_of_col_tl`.

```

6497   \group_begin:
6498   \tl_set:Nn \l_@@_type_of_col_tl c
6499   \keys_set_known:nnN { NiceMatrix / Auto } { #3, #5, #7 } \l_tmpa_tl
6500   \use:x
6501   {
6502     \exp_not:N \begin { NiceArrayWithDelims } { #1 } { #2 }
6503     { * { \int_use:N \l_@@_nb_cols_int } { \l_@@_type_of_col_tl } }
6504     [ \exp_not:N \l_tmpa_tl ]
6505   }
6506   \int_compare:nNnT \l_@@_first_row_int = 0
6507   {
6508     \int_compare:nNnT \l_@@_first_col_int = 0 { & }
6509     \prg_replicate:nn { \l_@@_nb_cols_int - 1 } { & }
6510     \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
6511   }
6512   \prg_replicate:nn \l_@@_nb_rows_int
6513   {
6514     \int_compare:nNnT \l_@@_first_col_int = 0 { & }

```

We put `{ }` before `#6` to avoid a hasty expansion of a potential `\arabic{iRow}` at the beginning of the row which would result in an incorrect value of that `iRow` (since `iRow` is incremented in the first cell of the row of the `\halign`).

```

6515     \prg_replicate:nn { \l_@@_nb_cols_int - 1 } { { } #6 & } #6
6516     \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
6517   }
6518   \int_compare:nNnT \l_@@_last_row_int > { -2 }
6519   {
6520     \int_compare:nNnT \l_@@_first_col_int = 0 { & }
6521     \prg_replicate:nn { \l_@@_nb_cols_int - 1 } { & }
6522     \int_compare:nNnT \l_@@_last_col_int > { -1 } { & } \\
6523   }
6524   \end { NiceArrayWithDelims }
6525   \group_end:
6526 }
6527 \cs_set_protected:Npn \@@_define_com:nnn #1 #2 #3
6528 {
6529   \cs_set_protected:cpn { #1 AutoNiceMatrix }
6530   {
6531     \str_gset:Nx \g_@@_name_env_str { #1 AutoNiceMatrix }
6532     \AutoNiceMatrixWithDelims { #2 } { #3 }
6533   }
6534 }
6535 \@@_define_com:nnn p ( )
6536 \@@_define_com:nnn b [ ]
6537 \@@_define_com:nnn v | |
6538 \@@_define_com:nnn V \ | \ |
6539 \@@_define_com:nnn B \{ \}

```

We define also a command `\AutoNiceMatrix` similar to the environment `{NiceMatrix}`.

```

6540 \NewDocumentCommand \AutoNiceMatrix { O { } m O { } m ! O { } }
6541 {
6542   \group_begin:

```

```

6543 \bool_set_true:N \l_@@_NiceArray_bool
6544 \AutoNiceMatrixWithDelims . . { #2 } { #4 } [ #1 , #3 , #5 ]
6545 \group_end:
6546 }

```

The redefinition of the command `\dotfill`

```

6547 \cs_set_eq:NN \@@_old_dotfill \dotfill
6548 \cs_new_protected:Npn \@@_dotfill:
6549 {

```

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}`).

```

6550 \@@_old_dotfill
6551 \bool_if:NT \l_@@_NiceTabular_bool
6552 { \group_insert_after:N \@@_dotfill_ii: }
6553 { \group_insert_after:N \@@_dotfill_i: }
6554 }
6555 \cs_new_protected:Npn \@@_dotfill_i: { \group_insert_after:N \@@_dotfill_ii: }
6556 \cs_new_protected:Npn \@@_dotfill_ii: { \group_insert_after:N \@@_dotfill_iii: }

```

Now, if the box is not empty (unfortunately, we can’t actually test whether the box is empty and that’s why we only consider it’s width), we insert `\@@_dotfill` (which is the saved version of `\dotfill`) in the cell of the array, and it will extend, since it is no longer in `\l_@@_cell_box`.

```

6557 \cs_new_protected:Npn \@@_dotfill_iii:
6558 { \dim_compare:nNnT { \box_wd:N \l_@@_cell_box } = \c_zero_dim \@@_old_dotfill }

```

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.

```

6559 \cs_new_protected:Npn \@@_diagbox:nn #1 #2
6560 {
6561 \tl_gput_right:Nx \g_@@_internal_code_after_tl
6562 {
6563 \@@_actually_diagbox:nnnnnn
6564 { \int_use:N \c@iRow }
6565 { \int_use:N \c@jCol }
6566 { \int_use:N \c@iRow }
6567 { \int_use:N \c@jCol }
6568 { \exp_not:n { #1 } }
6569 { \exp_not:n { #2 } }
6570 }

```

We put the cell with `\diagbox` in the sequence `\g_@@_pos_of_blocks_seq` because a cell with `\diagbox` must be considered as non empty by the key `corners`.

```

6571 \seq_gput_right:Nx \g_@@_pos_of_blocks_seq
6572 {
6573 { \int_use:N \c@iRow }
6574 { \int_use:N \c@jCol }
6575 { \int_use:N \c@iRow }
6576 { \int_use:N \c@jCol }

```

The last argument is for the name of the block.

```

6577 { }
6578 }
6579 }

```

The command `\diagbox` is also redefined locally when we draw a block.

The first four arguments of `\@@_actually_diagbox:nnnnnn` correspond to the rectangle (=block) to slash (we recall that it’s possible to use `\diagbox` in a `\Block`). The other two are the elements to draw below and above the diagonal line.

```

6580 \cs_new_protected:Npn \@@_actually_diagbox:nnnnnn #1 #2 #3 #4 #5 #6

```



```

6581 {
6582   \pgfpicture
6583   \pgf@relevantforpicturesizefalse
6584   \pgfrememberpicturepositiononpagetrue
6585   \@@_qpoint:n { row - #1 }
6586   \dim_set_eq:NN \l_tmpa_dim \pgf@y
6587   \@@_qpoint:n { col - #2 }
6588   \dim_set_eq:NN \l_tmpb_dim \pgf@x
6589   \pgfpathmoveto { \pgfpoint \l_tmpb_dim \l_tmpa_dim }
6590   \@@_qpoint:n { row - \@@_succ:n { #3 } }
6591   \dim_set_eq:NN \l_@@_tmpc_dim \pgf@y
6592   \@@_qpoint:n { col - \@@_succ:n { #4 } }
6593   \dim_set_eq:NN \l_@@_tmpd_dim \pgf@x
6594   \pgfpathlineto { \pgfpoint \l_@@_tmpd_dim \l_@@_tmpc_dim }
6595   {

```

The command `\CT@arc@` is a command of `colortbl` which sets the color of the rules in the array. The package `nicematrix` uses it even if `colortbl` is not loaded.

```

6596     \CT@arc@
6597     \pgfsetroundcap
6598     \pgfusepathqstroke
6599   }
6600   \pgfset { inner~sep = 1 pt }
6601   \pgfscope
6602   \pgftransformshift { \pgfpoint \l_tmpb_dim \l_@@_tmpc_dim }
6603   \pgfnode { rectangle } { south-west }
6604   {
6605     \begin { minipage } { 20 cm }
6606     \@@_math_toggle_token: #5 \@@_math_toggle_token:
6607     \end { minipage }
6608   }
6609   { }
6610   { }
6611 \endpgfscope
6612 \pgftransformshift { \pgfpoint \l_@@_tmpd_dim \l_tmpa_dim }
6613 \pgfnode { rectangle } { north-east }
6614 {
6615   \begin { minipage } { 20 cm }
6616   \raggedleft
6617   \@@_math_toggle_token: #6 \@@_math_toggle_token:
6618   \end { minipage }
6619 }
6620 { }
6621 { }
6622 \endpgfpicture
6623 }

```

The keyword `\CodeAfter`

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

```

6624 \keys_define:nn { NiceMatrix }
6625 {
6626   CodeAfter / rules .inherit:n = NiceMatrix / rules ,
6627   CodeAfter / sub-matrix .inherit:n = NiceMatrix / sub-matrix
6628 }
6629 \keys_define:nn { NiceMatrix / CodeAfter }
6630 {
6631   sub-matrix .code:n = \keys_set:nn { NiceMatrix / sub-matrix } { #1 } ,
6632   sub-matrix .value_required:n = true ,
6633   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,

```

```

6634     delimiters / color .value_required:n = true ,
6635     rules .code:n = \keys_set:nn { NiceMatrix / rules } { #1 } ,
6636     rules .value_required:n = true ,
6637     unknown .code:n = \@@_error:n { Unknown~key~for~CodeAfter }
6638 }

```

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. 123.

In the environments of `nicematrix`, `\CodeAfter` will be linked to `\@@_CodeAfter:`. That macro must *not* be protected since it begins with `\omit`.

```

6639 \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 `\`.

```

6640 \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`.

```

6641 \cs_new_protected:Npn \@@_CodeAfter_ii:n #1 \end
6642 {
6643     \tl_gput_right:Nn \g_nicematrix_code_after_tl { #1 }
6644     \@@_CodeAfter_iv:n
6645 }

```

We catch the argument of the command `\end` (in `#1`).

```

6646 \cs_new_protected:Npn \@@_CodeAfter_iv:n #1
6647 {

```

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.

```

6648     \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`.

```

6649     {
6650         \tl_gput_right:Nn \g_nicematrix_code_after_tl { \end { #1 } }
6651         \@@_CodeAfter_ii:n
6652     }
6653 }

```

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_@@_internal_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.

```

6654 \cs_new_protected:Npn \@@_delimiter:nnn #1 #2 #3
6655 {
6656     \pgfpicture
6657     \pgfrememberpicturepositiononpagetrue
6658     \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.

```

6659 \@@_qpoint:n { row - 1 }
6660 \dim_set_eq:NN \l_@@_y_initial_dim \pgf@y
6661 \@@_qpoint:n { row - \@@_succ:n \c@iRow }
6662 \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).

```

6663 \bool_if:nTF { #3 }
6664 { \dim_set_eq:NN \l_tmpa_dim \c_max_dim }
6665 { \dim_set:Nn \l_tmpa_dim { - \c_max_dim } }
6666 \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int
6667 {
6668   \cs_if_exist:cT
6669   { pgf @ sh @ ns @ \@@_env: - ##1 - #2 }
6670   {
6671     \pgfpointanchor
6672     { \@@_env: - ##1 - #2 }
6673     { \bool_if:nTF { #3 } { west } { east } }
6674     \dim_set:Nn \l_tmpa_dim
6675     { \bool_if:nTF { #3 } \dim_min:nn \dim_max:nn \l_tmpa_dim \pgf@x }
6676   }
6677 }

```

Now we can put the delimiter with a node of PGF.

```

6678 \pgfset { inner~sep = \c_zero_dim }
6679 \dim_zero:N \nulldelimiterspace
6680 \pgftransformshift
6681 {
6682   \pgfpoint
6683   { \l_tmpa_dim }
6684   { ( \l_@@_y_initial_dim + \l_@@_y_final_dim + \arrayrulewidth ) / 2 }
6685 }
6686 \pgfnode
6687 { rectangle }
6688 { \bool_if:nTF { #3 } { east } { west } }
6689 {

```

Here is the content of the PGF node, that is to say the delimiter, constructed with its right size.

```

6690 \nullfont
6691 \c_math_toggle_token
6692 \tl_if_empty:NF \l_@@_delimiters_color_tl
6693 { \color { \l_@@_delimiters_color_tl } }
6694 \bool_if:nTF { #3 } { \left #1 } { \left . }
6695 \vcenter
6696 {
6697   \nullfont
6698   \hrule \@height
6699   \dim_eval:n { \l_@@_y_initial_dim - \l_@@_y_final_dim }
6700   \@depth \c_zero_dim
6701   \@width \c_zero_dim
6702 }
6703 \bool_if:nTF { #3 } { \right . } { \right #1 }
6704 \c_math_toggle_token
6705 }
6706 { }
6707 { }
6708 \endpgfpicture
6709 }

```

The command `\SubMatrix`

```

6710 \keys_define:nn { NiceMatrix / sub-matrix }
6711 {
6712   extra-height .dim_set:N = \l_@@_submatrix_extra_height_dim ,
6713   extra-height .value_required:n = true ,
6714   left-xshift .dim_set:N = \l_@@_submatrix_left_xshift_dim ,
6715   left-xshift .value_required:n = true ,
6716   right-xshift .dim_set:N = \l_@@_submatrix_right_xshift_dim ,
6717   right-xshift .value_required:n = true ,
6718   xshift .meta:n = { left-xshift = #1, right-xshift = #1 } ,
6719   xshift .value_required:n = true ,
6720   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
6721   delimiters / color .value_required:n = true ,
6722   slim .bool_set:N = \l_@@_submatrix_slim_bool ,
6723   slim .default:n = true ,
6724   hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
6725   hlines .default:n = all ,
6726   vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
6727   vlines .default:n = all ,
6728   hvlines .meta:n = { hlines, vlines } ,
6729   hvlines .value_forbidden:n = true ,
6730 }
6731 \keys_define:nn { NiceMatrix }
6732 {
6733   SubMatrix .inherit:n = NiceMatrix / sub-matrix ,
6734   CodeAfter / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
6735   NiceMatrix / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
6736   NiceArray / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
6737   pNiceArray / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
6738   NiceMatrixOptions / sub-matrix .inherit:n = NiceMatrix / sub-matrix ,
6739 }

```

The following keys set is for the command `\SubMatrix` itself (not the tuning of `\SubMatrix` that can be done elsewhere).

```

6740 \keys_define:nn { NiceMatrix / SubMatrix }
6741 {
6742   delimiters / color .tl_set:N = \l_@@_delimiters_color_tl ,
6743   delimiters / color .value_required:n = true ,
6744   hlines .clist_set:N = \l_@@_submatrix_hlines_clist ,
6745   hlines .default:n = all ,
6746   vlines .clist_set:N = \l_@@_submatrix_vlines_clist ,
6747   vlines .default:n = all ,
6748   hvlines .meta:n = { hlines, vlines } ,
6749   hvlines .value_forbidden:n = true ,
6750   name .code:n =
6751     \tl_if_empty:nTF { #1 }
6752     { \@@_error:n { Invalid~name~format } }
6753     {
6754       \regex_match:nnTF { \A[A-Za-z][A-Za-z0-9]*\Z } { #1 }
6755       {
6756         \seq_if_in:NnTF \g_@@_submatrix_names_seq { #1 }
6757         { \@@_error:nn { Duplicate~name~for~SubMatrix } { #1 } }
6758         {
6759           \str_set:Nn \l_@@_submatrix_name_str { #1 }
6760           \seq_gput_right:Nn \g_@@_submatrix_names_seq { #1 }
6761         }
6762       }
6763       { \@@_error:n { Invalid~name~format } }
6764     } ,
6765   rules .code:n = \keys_set:nn { NiceMatrix / rules } { #1 } ,
6766   rules .value_required:n = true ,
6767   code .tl_set:N = \l_@@_code_tl ,
6768   code .value_required:n = true ,

```

```

6769     name .value_required:n = true ,
6770     unknown .code:n = \@@_error:n { Unknown-key-for-SubMatrix }
6771 }

6772 \NewDocumentCommand \@@_SubMatrix_in_code_before { m m m m ! 0 { } }
6773 {
6774     \peek_remove_spaces:n
6775     {
6776         \@@_cut_on_hyphen:w #3 \q_stop
6777         \tl_clear_new:N \l_@@_tmpc_tl
6778         \tl_clear_new:N \l_@@_tmpd_tl
6779         \tl_set_eq:NN \l_@@_tmpc_tl \l_tmpa_tl
6780         \tl_set_eq:NN \l_@@_tmpd_tl \l_tmpb_tl
6781         \@@_cut_on_hyphen:w #2 \q_stop
6782         \seq_gput_right:Nx \g_@@_submatrix_seq
6783         { { \l_tmpa_tl } { \l_tmpb_tl } { \l_@@_tmpc_tl } { \l_@@_tmpd_tl } }
6784         \tl_gput_right:Nn \g_@@_internal_code_after_tl
6785         { \SubMatrix { #1 } { #2 } { #3 } { #4 } [ #5 ] }
6786     }
6787 }

```

In the internal 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`.

```

6788 \hook_gput_code:nnn { begindocument } { . }
6789 {
6790     \tl_set:Nn \l_@@_argspec_tl { m m m m 0 { } E { _ ^ } { { } { } } }
6791     \tl_set_rescan:Nno \l_@@_argspec_tl { } \l_@@_argspec_tl
6792     \exp_args:NNV \NewDocumentCommand \@@_SubMatrix \l_@@_argspec_tl
6793     {
6794         \peek_remove_spaces:n
6795         {
6796             \@@_sub_matrix:nnnnnnn
6797             { #1 } { #2 } { #3 } { #4 } { #5 } { #6 } { #7 }
6798         }
6799     }
6800 }

```

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).

```

6801 \cs_new_protected:Npn \@@_compute_i_j:nn #1 #2
6802 {
6803     \tl_clear_new:N \l_@@_first_i_tl
6804     \tl_clear_new:N \l_@@_first_j_tl
6805     \tl_clear_new:N \l_@@_last_i_tl
6806     \tl_clear_new:N \l_@@_last_j_tl
6807     \@@_cut_on_hyphen:w #1 \q_stop

```

```

6808 \tl_if_eq:NnTF \l_tmpa_tl { last }
6809   { \tl_set:NV \l_@@_first_i_tl \c@iRow }
6810   { \tl_set_eq:NN \l_@@_first_i_tl \l_tmpa_tl }
6811 \tl_if_eq:NnTF \l_tmpb_tl { last }
6812   { \tl_set:NV \l_@@_first_j_tl \c@jCol }
6813   { \tl_set_eq:NN \l_@@_first_j_tl \l_tmpb_tl }
6814 \@@_cut_on_hyphen:w #2 \q_stop
6815 \tl_if_eq:NnTF \l_tmpa_tl { last }
6816   { \tl_set:NV \l_@@_last_i_tl \c@iRow }
6817   { \tl_set_eq:NN \l_@@_last_i_tl \l_tmpa_tl }
6818 \tl_if_eq:NnTF \l_tmpb_tl { last }
6819   { \tl_set:NV \l_@@_last_j_tl \c@jCol }
6820   { \tl_set_eq:NN \l_@@_last_j_tl \l_tmpb_tl }
6821 }

6822 \cs_new_protected:Npn \@@_sub_matrix:nnnnnnn #1 #2 #3 #4 #5 #6 #7
6823 {
6824   \group_begin:

```

The four following token lists correspond to the position of the \SubMatrix.

```

6825 \@@_compute_i_j:nn { #2 } { #3 }
6826 \bool_lazy_or:nnTF
6827   { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
6828   { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
6829   { \@@_error:nn { Construct-too-large } { \SubMatrix } }
6830 {
6831   \str_clear_new:N \l_@@_submatrix_name_str
6832   \keys_set:nn { NiceMatrix / SubMatrix } { #5 }
6833   \pgfpicture
6834   \pgfrememberpicturepositiononpagetrue
6835   \pgf@relevantforpicturesizefalse
6836   \pgfset { inner-sep = \c_zero_dim }
6837   \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
6838   \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }

```

The last value of \int_step_inline:nnn is provided by curriification.

```

6839 \bool_if:NnTF \l_@@_submatrix_slim_bool
6840   { \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl }
6841   { \int_step_inline:nnn \l_@@_first_row_int \g_@@_row_total_int }
6842   {
6843     \cs_if_exist:cT
6844       { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
6845       {
6846         \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
6847         \dim_set:Nn \l_@@_x_initial_dim
6848           { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
6849       }
6850     \cs_if_exist:cT
6851       { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
6852       {
6853         \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
6854         \dim_set:Nn \l_@@_x_final_dim
6855           { \dim_max:nn \l_@@_x_final_dim \pgf@x }
6856       }
6857   }
6858   \dim_compare:nNnTF \l_@@_x_initial_dim = \c_max_dim
6859     { \@@_error:nn { impossible-delimiter } { left } }
6860     {
6861       \dim_compare:nNnTF \l_@@_x_final_dim = { - \c_max_dim }
6862         { \@@_error:nn { impossible-delimiter } { right } }
6863         { \@@_sub_matrix_i:nnnn { #1 } { #4 } { #6 } { #7 } }
6864     }
6865   \endpgfpicture
6866 }
6867 \group_end:

```

```
6868 }
```

#1 is the left delimiter, #2 is the right one, #3 is the subscript and #4 is the superscript.

```
6869 \cs_new_protected:Npn \@@_sub_matrix_i:nnnn #1 #2 #3 #4
6870 {
6871   \@@_qpoint:n { row - \l_@@_first_i_tl - base }
6872   \dim_set:Nn \l_@@_y_initial_dim
6873     { \pgf@y + ( \box_ht:N \strutbox + \extrarowheight ) * \arraystretch }
6874   \@@_qpoint:n { row - \l_@@_last_i_tl - base }
6875   \dim_set:Nn \l_@@_y_final_dim
6876     { \pgf@y - ( \box_dp:N \strutbox ) * \arraystretch }
6877   \int_step_inline:nnn \l_@@_first_col_int \g_@@_col_total_int
6878     {
6879     \cs_if_exist:cT
6880       { \pgf @ sh @ ns @ \@@_env: - \l_@@_first_i_tl - ##1 }
6881       {
6882         \pgfpointanchor { \@@_env: - \l_@@_first_i_tl - ##1 } { north }
6883         \dim_set:Nn \l_@@_y_initial_dim
6884           { \dim_max:nn \l_@@_y_initial_dim \pgf@y }
6885       }
6886     \cs_if_exist:cT
6887       { \pgf @ sh @ ns @ \@@_env: - \l_@@_last_i_tl - ##1 }
6888       {
6889         \pgfpointanchor { \@@_env: - \l_@@_last_i_tl - ##1 } { south }
6890         \dim_set:Nn \l_@@_y_final_dim
6891           { \dim_min:nn \l_@@_y_final_dim \pgf@y }
6892       }
6893     }
6894   \dim_set:Nn \l_tmpa_dim
6895     {
6896     \l_@@_y_initial_dim - \l_@@_y_final_dim +
6897     \l_@@_submatrix_extra_height_dim - \arrayrulewidth
6898     }
6899   \dim_zero:N \nulldelimiterspace
```

We will draw the rules in the \SubMatrix.

```
6900 \group_begin:
6901 \pgfsetlinewidth { 1.1 \arrayrulewidth }
6902 \tl_if_empty:NF \l_@@_rules_color_tl
6903   { \exp_after:wN \@@_set_CT@arc@: \l_@@_rules_color_tl \q_stop }
6904 \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`.

```
6905 \seq_map_inline:Nn \g_@@_cols_vlism_seq
6906 {
6907   \int_compare:nNnT \l_@@_first_j_tl < { ##1 }
6908   {
6909     \int_compare:nNnT
6910       { ##1 } < { \int_eval:n { \l_@@_last_j_tl + 1 } }
6911     {
```

First, we extract the value of the abscissa of the rule we have to draw.

```
6912       \@@_qpoint:n { col - ##1 }
6913       \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
6914       \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
6915       \pgfusepathqstroke
6916     }
6917   }
6918 }
```

Now, we draw the vertical rules specified in the key `vlines` of `\SubMatrix`. The last argument of `\int_step_inline:nn` or `\clist_map_inline:Nn` is given by curryfication.

```

6919 \tl_if_eq:NnTF \l_@@_submatrix_vlines_clist { all }
6920 { \int_step_inline:nn { \l_@@_last_j_tl - \l_@@_first_j_tl } }
6921 { \clist_map_inline:Nn \l_@@_submatrix_vlines_clist }
6922 {
6923   \bool_lazy_and:nnTF
6924   { \int_compare_p:nNn { ##1 } > 0 }
6925   {
6926     \int_compare_p:nNn
6927     { ##1 } < { \l_@@_last_j_tl - \l_@@_first_j_tl + 1 } }
6928   {
6929     @@_qpoint:n { col - \int_eval:n { ##1 + \l_@@_first_j_tl } }
6930     \pgfpathmoveto { \pgfpoint \pgf@x \l_@@_y_initial_dim }
6931     \pgfpathlineto { \pgfpoint \pgf@x \l_@@_y_final_dim }
6932     \pgfusepathqstroke
6933   }
6934   { @@_error:nnn { Wrong~line~in~SubMatrix } { vertical } { ##1 } }
6935 }

```

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.

```

6936 \tl_if_eq:NnTF \l_@@_submatrix_hlines_clist { all }
6937 { \int_step_inline:nn { \l_@@_last_i_tl - \l_@@_first_i_tl } }
6938 { \clist_map_inline:Nn \l_@@_submatrix_hlines_clist }
6939 {
6940   \bool_lazy_and:nnTF
6941   { \int_compare_p:nNn { ##1 } > 0 }
6942   {
6943     \int_compare_p:nNn
6944     { ##1 } < { \l_@@_last_i_tl - \l_@@_first_i_tl + 1 } }
6945   {
6946     @@_qpoint:n { row - \int_eval:n { ##1 + \l_@@_first_i_tl } }

```

We use a group to protect `\l_tmpa_dim` and `\l_tmpb_dim`.

```

6947 \group_begin:

```

We compute in `\l_tmpa_dim` the x -value of the left end of the rule.

```

6948 \dim_set:Nn \l_tmpa_dim
6949 { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
6950 \str_case:nn { #1 }
6951 {
6952   ( { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
6953   [ { \dim_sub:Nn \l_tmpa_dim { 0.2 mm } }
6954   \{ { \dim_sub:Nn \l_tmpa_dim { 0.9 mm } }
6955   }
6956   \pgfpathmoveto { \pgfpoint \l_tmpa_dim \pgf@y }

```

We compute in `\l_tmpb_dim` the x -value of the right end of the rule.

```

6957 \dim_set:Nn \l_tmpb_dim
6958 { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
6959 \str_case:nn { #2 }
6960 {
6961   ) { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
6962   ] { \dim_add:Nn \l_tmpb_dim { 0.2 mm } }
6963   \} { \dim_add:Nn \l_tmpb_dim { 0.9 mm } }
6964 }
6965 \pgfpathlineto { \pgfpoint \l_tmpb_dim \pgf@y }
6966 \pgfusepathqstroke
6967 \group_end:
6968 }
6969 { @@_error:nnn { Wrong~line~in~SubMatrix } { horizontal } { ##1 } }
6970 }

```


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).

```

6971 \str_if_empty:NF \l_@@_submatrix_name_str
6972 {
6973   \@@_pgf_rect_node:nnnnn \l_@@_submatrix_name_str
6974   \l_@@_x_initial_dim \l_@@_y_initial_dim
6975   \l_@@_x_final_dim \l_@@_y_final_dim
6976 }
6977 \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`.

```

6978 \begin { pgfscope }
6979 \pgftransformshift
6980 {
6981   \pgfpoint
6982   { \l_@@_x_initial_dim - \l_@@_submatrix_left_xshift_dim }
6983   { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
6984 }
6985 \str_if_empty:NTF \l_@@_submatrix_name_str
6986 { \@@_node_left:nn #1 { } }
6987 { \@@_node_left:nn #1 { \@@_env: - \l_@@_submatrix_name_str - left } }
6988 \end { pgfscope }

```

Now, we deal with the right delimiter.

```

6989 \pgftransformshift
6990 {
6991   \pgfpoint
6992   { \l_@@_x_final_dim + \l_@@_submatrix_right_xshift_dim }
6993   { ( \l_@@_y_initial_dim + \l_@@_y_final_dim ) / 2 }
6994 }
6995 \str_if_empty:NTF \l_@@_submatrix_name_str
6996 { \@@_node_right:nnnn #2 { } { #3 } { #4 } }
6997 {
6998   \@@_node_right:nnnn #2
6999   { \@@_env: - \l_@@_submatrix_name_str - right } { #3 } { #4 }
7000 }
7001 \cs_set_eq:NN \pgfpointanchor \@@_pgfpointanchor:n
7002 \flag_clear_new:n { nicematrix }
7003 \l_@@_code_tl
7004 }

```

In the key code of the command `\SubMatrix` there may be Tikz instructions. We want that, in these instructions, the i and j in specifications of nodes of the forms $i-j$, $\text{row-}i$, $\text{col-}j$ and $i-|j$ refer to the number of row and column *relative* of the current `\SubMatrix`. That's why we will patch (locally in the `\SubMatrix`) the command `\pgfpointanchor`.

```

7005 \cs_set_eq:NN \@@_old_pgfpntanchor \pgfpointanchor

```

The following command will be linked to `\pgfpointanchor` just before the execution of the option code of the command `\SubMatrix`. In this command, we catch the argument #1 of `\pgfpointanchor` and we apply to it the command `\@@_pgfpointanchor_i:nn` before passing it to the original `\pgfpointanchor`. We have to act in an expandable way because the command `\pgfpointanchor` is used in names of Tikz nodes which are computed in an expandable way.

```

7006 \cs_new_protected:Npn \@@_pgfpointanchor:n #1
7007 {
7008   \use:e
7009   { \exp_not:N \@@_old_pgfpntanchor { \@@_pgfpointanchor_i:nn #1 } }
7010 }

```

In fact, the argument of `\pgfpointanchor` is always of the form `\a_command { name_of_node }` where “name_of_node” is the name of the Tikz node without the potential prefix and suffix. That’s why we catch two arguments and work only on the second by trying (first) to extract an hyphen -.

```

7011 \cs_new:Npn \@@_pgfpointanchor_i:nn #1 #2
7012   { #1 { \@@_pgfpointanchor_ii:w #2 - \q_stop } }

```

Since `\seq_if_in:NnTF` and `\clist_if_in:NnTF` are not expandable, we will use the following token list and `\str_case:nVTF` to test whether we have an integer or not.

```

7013 \tl_const:Nn \c_@@_integers_alist_tl
7014   {
7015     { 1 } { } { 2 } { } { 3 } { } { 4 } { } { 5 } { }
7016     { 6 } { } { 7 } { } { 8 } { } { 9 } { } { 10 } { }
7017     { 11 } { } { 12 } { } { 13 } { } { 14 } { } { 15 } { }
7018     { 16 } { } { 17 } { } { 18 } { } { 19 } { } { 20 } { }
7019   }

```

```

7020 \cs_new:Npn \@@_pgfpointanchor_ii:w #1-#2\q_stop
7021   {

```

If there is no hyphen, that means that the node is of the form of a single number (ex.: 5 or 11). In that case, we are in an analysis which result from a specification of node of the form $i-j$. In that case, the i of the number of row arrives first (and alone) in a `\pgfpointanchor` and, the, the j arrives (alone) in the following `\pgfpointanchor`. In order to know whether we have a number of row or a number of column, we keep track of the number of such treatments by the expandable flag called `nicematrix`.

```

7022   \tl_if_empty:nTF { #2 }
7023   {
7024     \str_case:nVTF { #1 } \c_@@_integers_alist_tl
7025     {
7026       \flag_raise:n { nicematrix }
7027       \int_if_even:nTF { \flag_height:n { nicematrix } }
7028       { \int_eval:n { #1 + \l_@@_first_i_tl - 1 } }
7029       { \int_eval:n { #1 + \l_@@_first_j_tl - 1 } }
7030     }
7031     { #1 }
7032   }

```

If there is an hyphen, we have to see whether we have a node of the form $i-j$, row- i or col- j .

```

7033   { \@@_pgfpointanchor_iii:w { #1 } #2 }
7034   }

```

There was an hyphen in the name of the node and that’s why we have to retrieve the extra hyphen we have put (cf. `\@@_pgfpointanchor_i:nn`).

```

7035 \cs_new:Npn \@@_pgfpointanchor_iii:w #1 #2 -
7036   {
7037     \str_case:nnF { #1 }
7038     {
7039       { row } { row - \int_eval:n { #2 + \l_@@_first_i_tl - 1 } }
7040       { col } { col - \int_eval:n { #2 + \l_@@_first_j_tl - 1 } }
7041     }

```

Now the case of a node of the form $i-j$.

```

7042   {
7043     \int_eval:n { #1 + \l_@@_first_i_tl - 1 }
7044     - \int_eval:n { #2 + \l_@@_first_j_tl - 1 }
7045   }
7046   }

```

The command `\@@_node_left:nn` puts the left delimiter with the correct size. The argument `#1` is the delimiter to put. The argument `#2` is the name we will give to this PGF node (if the key `name` has been used in `\SubMatrix`).

```

7047 \cs_new_protected:Npn \@@_node_left:nn #1 #2
7048 {
7049   \pgfnode
7050   { rectangle }
7051   { east }
7052   {
7053     \nullfont
7054     \c_math_toggle_token
7055     \tl_if_empty:NF \l_@@_delimiters_color_tl
7056     { \color { \l_@@_delimiters_color_tl } }
7057     \left #1
7058     \vcenter
7059     {
7060       \nullfont
7061       \hrule \@height \l_tmpa_dim
7062       \@depth \c_zero_dim
7063       \@width \c_zero_dim
7064     }
7065     \right .
7066     \c_math_toggle_token
7067   }
7068   { #2 }
7069   { }
7070 }

```

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.

```

7071 \cs_new_protected:Npn \@@_node_right:nnnn #1 #2 #3 #4
7072 {
7073   \pgfnode
7074   { rectangle }
7075   { west }
7076   {
7077     \nullfont
7078     \c_math_toggle_token
7079     \tl_if_empty:NF \l_@@_delimiters_color_tl
7080     { \color { \l_@@_delimiters_color_tl } }
7081     \left .
7082     \vcenter
7083     {
7084       \nullfont
7085       \hrule \@height \l_tmpa_dim
7086       \@depth \c_zero_dim
7087       \@width \c_zero_dim
7088     }
7089     \right #1
7090     \tl_if_empty:NF { #3 } { _ { \smash { #3 } } }
7091     ^ { \smash { #4 } }
7092     \c_math_toggle_token
7093   }
7094   { #2 }
7095   { }
7096 }

```

Les commandes `\UnderBrace` et `\OverBrace`

The following commands will be linked to `\UnderBrace` and `\OverBrace` in the `\CodeAfter`.

```

7097 \NewDocumentCommand \@@_UnderBrace { 0 { } m m m 0 { } }
7098 {
7099   \peek_remove_spaces:n
7100   { \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { under } }
7101 }
7102 \NewDocumentCommand \@@_OverBrace { 0 { } m m m 0 { } }
7103 {
7104   \peek_remove_spaces:n
7105   { \@@_brace:nnnnn { #2 } { #3 } { #4 } { #1 , #5 } { over } }
7106 }
7107 \keys_define:nn { NiceMatrix / Brace }
7108 {
7109   left-shorten .bool_set:N = \l_@@_brace_left_shorten_bool ,
7110   left-shorten .default:n = true ,
7111   right-shorten .bool_set:N = \l_@@_brace_right_shorten_bool ,
7112   shorten .meta:n = { left-shorten , right-shorten } ,
7113   right-shorten .default:n = true ,
7114   yshift .dim_set:N = \l_@@_brace_yshift_dim ,
7115   yshift .value_required:n = true ,
7116   yshift .initial:n = \c_zero_dim ,
7117   unknown .code:n = \@@_error:n { Unknown-key-for-Brace }
7118 }

```

#1 is the first cell of the rectangle (with the syntax $i-lj$; #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*.

```

7119 \cs_new_protected:Npn \@@_brace:nnnnn #1 #2 #3 #4 #5
7120 {
7121   \group_begin:

```

The four following token lists correspond to the position of the sub-matrix to which a brace will be attached.

```

7122 \@@_compute_i_j:nn { #1 } { #2 }
7123 \bool_lazy_or:nnTF
7124 { \int_compare_p:nNn \l_@@_last_i_tl > \g_@@_row_total_int }
7125 { \int_compare_p:nNn \l_@@_last_j_tl > \g_@@_col_total_int }
7126 {
7127   \str_if_eq:nnTF { #5 } { under }
7128   { \@@_error:nn { Construct-too-large } { \UnderBrace } }
7129   { \@@_error:nn { Construct-too-large } { \OverBrace } }
7130 }
7131 {
7132   \keys_set:nn { NiceMatrix / Brace } { #4 }
7133   \pgfpicture
7134   \pgfrememberpicturepositiononpagetrue
7135   \pgf@relevantforpicturesizefalse
7136   \bool_if:NT \l_@@_brace_left_shorten_bool
7137   {
7138     \dim_set_eq:NN \l_@@_x_initial_dim \c_max_dim
7139     \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
7140     {
7141       \cs_if_exist:cT
7142       { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_first_j_tl }
7143       {
7144         \pgfpointanchor { \@@_env: - ##1 - \l_@@_first_j_tl } { west }
7145         \dim_set:Nn \l_@@_x_initial_dim
7146         { \dim_min:nn \l_@@_x_initial_dim \pgf@x }
7147       }
7148     }
7149   }
7150   \bool_lazy_or:nnT
7151   { \bool_not_p:n \l_@@_brace_left_shorten_bool }
7152   { \dim_compare_p:nNn \l_@@_x_initial_dim = \c_max_dim }

```

```

7153     {
7154         \@@_qpoint:n { col - \l_@@_first_j_tl }
7155         \dim_set_eq:NN \l_@@_x_initial_dim \pgf@x
7156     }
7157     \bool_if:NT \l_@@_brace_right_shorten_bool
7158     {
7159         \dim_set:Nn \l_@@_x_final_dim { - \c_max_dim }
7160         \int_step_inline:nnn \l_@@_first_i_tl \l_@@_last_i_tl
7161         {
7162             \cs_if_exist:cT
7163             { pgf @ sh @ ns @ \@@_env: - ##1 - \l_@@_last_j_tl }
7164             {
7165                 \pgfpointanchor { \@@_env: - ##1 - \l_@@_last_j_tl } { east }
7166                 \dim_set:Nn \l_@@_x_final_dim
7167                 { \dim_max:nn \l_@@_x_final_dim \pgf@x }
7168             }
7169         }
7170     }
7171     \bool_lazy_or:nnT
7172     { \bool_not_p:n \l_@@_brace_right_shorten_bool }
7173     { \dim_compare_p:nNn \l_@@_x_final_dim = { - \c_max_dim } }
7174     {
7175         \@@_qpoint:n { col - \int_eval:n { \l_@@_last_j_tl + 1 } }
7176         \dim_set_eq:NN \l_@@_x_final_dim \pgf@x
7177     }
7178     \pgfset { inner-sep = \c_zero_dim }
7179     \str_if_eq:nnTF { #5 } { under }
7180     { \@@_underbrace_i:n { #3 } }
7181     { \@@_overbrace_i:n { #3 } }
7182     \endpgfpicture
7183 }
7184 \group_end:
7185 }

```

The argument is the text to put above the brace.

```

7186 \cs_new_protected:Npn \@@_overbrace_i:n #1
7187 {
7188     \@@_qpoint:n { row - \l_@@_first_i_tl }
7189     \pgftransformshift
7190     {
7191         \pgfpoint
7192         { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
7193         { \pgf@y + \l_@@_brace_yshift_dim }
7194     }
7195     \pgfnode
7196     { rectangle }
7197     { south }
7198     {
7199         \vbox_top:n
7200         {
7201             \group_begin:
7202             \everycr { }
7203             \halign
7204             {
7205                 \hfil ## \hfil \cr
7206                 \@@_math_toggle_token: #1 \@@_math_toggle_token: \cr
7207                 \noalign { \skip_vertical:n { 4.5 pt } \nointerlineskip }
7208                 \hbox_to_wd:nn
7209                 { \l_@@_x_final_dim - \l_@@_x_initial_dim }
7210                 { \downbracefill } \cr
7211             }
7212             \group_end:
7213         }
7214     }

```

```

7215     { }
7216     { }
7217 }

```

The argument is the text to put under the brace.

```

7218 \cs_new_protected:Npn \@@_underbrace_i:n #1
7219 {
7220   \@@_qpoint:n { row - \int_eval:n { \l_@@_last_i_tl + 1 } }
7221   \pgftransformshift
7222   {
7223     \pgfpoint
7224     { ( \l_@@_x_initial_dim + \l_@@_x_final_dim ) / 2 }
7225     { \pgf@y - \l_@@_brace_yshift_dim }
7226   }
7227   \pgfnode
7228   { rectangle }
7229   { north }
7230   {
7231     \group_begin:
7232     \everycr { }
7233     \vbox:n
7234     {
7235       \halign
7236       {
7237         \hfil ## \hfil \crrc
7238         \hbox_to_wd:nn
7239         { \l_@@_x_final_dim - \l_@@_x_initial_dim }
7240         { \upbracefill } \cr
7241         \noalign { \skip_vertical:n { 4.5 pt } \nointerlineskip }
7242         \@@_math_toggle_token: #1 \@@_math_toggle_token: \cr
7243       }
7244     }
7245     \group_end:
7246   }
7247   { }
7248   { }
7249 }

```

We process the options at package loading

We process the options when the package is loaded (with `\usepackage`) but we recommend to use `\NiceMatrixOptions` instead.

We must process these options after the definition of the environment `{NiceMatrix}` because the option `renew-matrix` executes the code `\cs_set_eq:NN \env@matrix \NiceMatrix`.

Of course, the command `\NiceMatrix` must be defined before such an instruction is executed.

The boolean `\g_@@_footnotehyper_bool` will indicate if the option `footnotehyper` is used.

```

7250 \bool_new:N \c_@@_footnotehyper_bool

```

The boolean `\c_@@_footnote_bool` will indicate if the option `footnote` is used, but quickly, it will also be set to true if the option `footnotehyper` is used.

```

7251 \bool_new:N \c_@@_footnote_bool

7252 \@@_msg_new:nnn { Unknown~key~for~package }
7253 {
7254   The~key~'\l_keys_key_str'~is~unknown. \\
7255   If~you~go~on,~it~will~be~ignored. \\
7256   For~a~list~of~the~available~keys,~type~H~<return>.
7257 }
7258 {
7259   The~available~keys~are~(in~alphabetic~order):~

```

```

7260 footnote,~
7261 footnotehyper,~
7262 renew-dots,~and
7263 renew-matrix.
7264 }

7265 \keys_define:nn { NiceMatrix / Package }
7266 {
7267   renew-dots .bool_set:N = \l_@@_renew_dots_bool ,
7268   renew-dots .value_forbidden:n = true ,
7269   renew-matrix .code:n = \@@_renew_matrix: ,
7270   renew-matrix .value_forbidden:n = true ,
7271   transparent .code:n = \@@_fatal:n { Key-transparent } ,
7272   transparent .value_forbidden:n = true,
7273   footnote .bool_set:N = \c_@@_footnote_bool ,
7274   footnotehyper .bool_set:N = \c_@@_footnotehyper_bool ,
7275   unknown .code:n = \@@_error:n { Unknown-key-for-package }
7276 }
7277 \ProcessKeysOptions { NiceMatrix / Package }

7278 \@@_msg_new:nn { footnote-with-footnotehyper-package }
7279 {
7280   You~can't~use~the~option~'footnote'~because~the~package~
7281   footnotehyper~has~already~been~loaded.~
7282   If~you~want,~you~can~use~the~option~'footnotehyper'~and~the~footnotes~
7283   within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
7284   of~the~package~footnotehyper.\\
7285   If~you~go~on,~the~package~footnote~won't~be~loaded.
7286 }

7287 \@@_msg_new:nn { footnotehyper-with-footnote-package }
7288 {
7289   You~can't~use~the~option~'footnotehyper'~because~the~package~
7290   footnote~has~already~been~loaded.~
7291   If~you~want,~you~can~use~the~option~'footnote'~and~the~footnotes~
7292   within~the~environments~of~nicematrix~will~be~extracted~with~the~tools~
7293   of~the~package~footnote.\\
7294   If~you~go~on,~the~package~footnotehyper~won't~be~loaded.
7295 }

7296 \bool_if:NT \c_@@_footnote_bool
7297 {

```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```

7298 \@ifclassloaded { beamer }
7299 { \bool_set_false:N \c_@@_footnote_bool }
7300 {
7301   \@ifpackageloaded { footnotehyper }
7302   { \@@_error:n { footnote-with-footnotehyper-package } }
7303   { \usepackage { footnote } }
7304 }
7305 }

7306 \bool_if:NT \c_@@_footnotehyper_bool
7307 {

```

The class beamer has its own system to extract footnotes and that's why we have nothing to do if beamer is used.

```

7308 \@ifclassloaded { beamer }
7309 { \bool_set_false:N \c_@@_footnote_bool }
7310 {
7311   \@ifpackageloaded { footnote }
7312   { \@@_error:n { footnotehyper-with-footnote-package } }

```

```

7313     { \usepackage { footnotehyper } }
7314   }
7315   \bool_set_true:N \c_@@_footnote_bool
7316 }

```

The flag `\c_@@_footnote_bool` is raised and so, we will only have to test `\c_@@_footnote_bool` in order to know if we have to insert an environment `{savenotes}`.

Error messages of the package

```

7317 \seq_new:N \c_@@_types_of_matrix_seq
7318 \seq_set_from_clist:Nn \c_@@_types_of_matrix_seq
7319 {
7320   NiceMatrix ,
7321   pNiceMatrix , bNiceMatrix , vNiceMatrix, BNiceMatrix, VNiceMatrix
7322 }
7323 \seq_set_map_x:NNn \c_@@_types_of_matrix_seq \c_@@_types_of_matrix_seq
7324 { \tl_to_str:n { #1 } }

```

If the user uses too much columns, the command `\@@_error_too_much_cols:` is executed. This command raises an error but try to give the best information to the user in the error message. The command `\seq_if_in:NVTF` is not expandable and that's why we can't put it in the error message itself. We have to do the test before the `\@@_fatal:n`.

```

7325 \cs_new_protected:Npn \@@_error_too_much_cols:
7326 {
7327   \seq_if_in:NVTF \c_@@_types_of_matrix_seq \g_@@_name_env_str
7328   {
7329     \int_compare:nNnTF \l_@@_last_col_int = { -2 }
7330     { \@@_fatal:n { too~much~cols~for~matrix } }
7331     {
7332       \bool_if:NF \l_@@_last_col_without_value_bool
7333       { \@@_fatal:n { too~much~cols~for~matrix~with~last~col } }
7334     }
7335   }
7336   { \@@_fatal:n { too~much~cols~for~array } }
7337 }

```

The following command must *not* be protected since it's used in an error message.

```

7338 \cs_new:Npn \@@_message_hdotsfor:
7339 {
7340   \tl_if_empty:VF \g_@@_HVdotsfor_lines_tl
7341   { ~Maybe~you~use~of~\token_to_str:N \Hdotsfor\ is~incorrect.}
7342 }
7343 \@@_msg_new:nn { negative~weight }
7344 {
7345   The~weight~of~the~'X'~columns~must~be~positive~and~you~have~used~
7346   the~value~'#1'.~If~you~go~on,~the~absolute~value~will~be~used.
7347 }
7348 \@@_msg_new:nn { too~much~cols~for~matrix~with~last~col }
7349 {
7350   You~try~to~use~more~columns~than~allowed~by~your~
7351   \@@_full_name_env:.\@@_message_hdotsfor:\ The~maximal~number~of~
7352   columns~is~\int_eval:n { \l_@@_last_col_int - 1 }~(plus~the~
7353   exterior~columns).~This~error~is~fatal.
7354 }
7355 \@@_msg_new:nn { too~much~cols~for~matrix }
7356 {
7357   You~try~to~use~more~columns~than~allowed~by~your~
7358   \@@_full_name_env:.\@@_message_hdotsfor:\ Recall~that~the~maximal~
7359   number~of~columns~for~a~matrix~is~fixed~by~the~LaTeX~counter~
7360   'MaxMatrixCols'.~Its~actual~value~is~\int_use:N \c@MaxMatrixCols.~

```



```

7361     This-error-is-fatal.
7362 }

```

For the following message, remind that the test is not done after the construction of the array but in each row. That's why we have to put \c@jCol-1 and not \c@jCol.

```

7363 \@@_msg_new:nn { too-much-cols-for-array }
7364 {
7365     You-try-to-use-more-columns-than-allowed-by-your-
7366     \@@_full_name_env:.\@@_message_hdotsfor:\ The-maximal-number-of-columns-is-
7367     \int_use:N \g_@@_static_num_of_col_int\
7368     ~(plus-the-potential-exterior-ones).~
7369     This-error-is-fatal.
7370 }
7371 \@@_msg_new:nn { last-col-not-used }
7372 {
7373     The-key~'last-col'~is~in~force~but~you~have~not~used~that~last~column~
7374     in~your~\@@_full_name_env:~.~However,~you~can~go~on.
7375 }
7376 \@@_msg_new:nn { columns-not-used }
7377 {
7378     The-preamble~of~your~\@@_full_name_env:\ announces~\int_use:N
7379     \g_@@_static_num_of_col_int\ columns~but~you~use~only~\int_use:N \c@jCol.\\
7380     You~can~go~on~but~the~columns~you~did~not~used~won't~be~created.
7381 }
7382 \@@_msg_new:nn { in-first-col }
7383 {
7384     You~can't~use~the~command~#1 in~the~first~column~(number~0)~of~the~array.\\
7385     If~you~go~on,~this~command~will~be~ignored.
7386 }
7387 \@@_msg_new:nn { in-last-col }
7388 {
7389     You~can't~use~the~command~#1 in~the~last~column~(exterior)~of~the~array.\\
7390     If~you~go~on,~this~command~will~be~ignored.
7391 }
7392 \@@_msg_new:nn { in-first-row }
7393 {
7394     You~can't~use~the~command~#1 in~the~first~row~(number~0)~of~the~array.\\
7395     If~you~go~on,~this~command~will~be~ignored.
7396 }
7397 \@@_msg_new:nn { in-last-row }
7398 {
7399     You~can't~use~the~command~#1 in~the~last~row~(exterior)~of~the~array.\\
7400     If~you~go~on,~this~command~will~be~ignored.
7401 }
7402 \@@_msg_new:nn { double-closing-delimiter }
7403 {
7404     You~can't~put~a~second~closing~delimiter~"#1"~just~after~a~first~closing~
7405     delimiter.~This~delimiter~will~be~ignored.
7406 }
7407 \@@_msg_new:nn { delimiter-after-opening }
7408 {
7409     You~can't~put~a~second~delimiter~"#1"~just~after~a~first~opening~
7410     delimiter.~This~delimiter~will~be~ignored.
7411 }
7412 \@@_msg_new:nn { bad-option-for-line-style }
7413 {
7414     Since~you~haven't~loaded~Tikz,~the~only~value~you~can~give~to~'line-style'~
7415     is~'standard'.~If~you~go~on,~this~key~will~be~ignored.
7416 }

```

```

7417 \@@_msg_new:nn { Unknown-key-for-xdots }
7418 {
7419   As-for-now,~there-is-only~three-keys~available~here:~'color',~'line-style'~
7420   and~'shorten'~(and-you-try-to-use~'\l_keys_key_str')~.~If-you-go-on,~
7421   this-key-will-be-ignored.
7422 }
7423 \@@_msg_new:nn { Unknown-key-for-rowcolors }
7424 {
7425   As-for-now,~there-is-only~two-keys~available~here:~'cols'~and~'respect-blocks'~
7426   (and-you-try-to-use~'\l_keys_key_str')~.~If-you-go-on,~
7427   this-key-will-be-ignored.
7428 }
7429 \@@_msg_new:nn { ampersand-in-light-syntax }
7430 {
7431   You-can't-use-an-ampersand~(\token_to_str:N &)~to-separate-columns~because~
7432   ~you-have-used~the~key~'light-syntax'~.~This-error-is-fatal.
7433 }
7434 \@@_msg_new:nn { Construct-too-large }
7435 {
7436   Your-command~\token_to_str:N #1
7437   can't-be-drawn~because~your~matrix-is-too-small.\\
7438   If-you-go-on,~this-command-will-be-ignored.
7439 }
7440 \@@_msg_new:nn { double-backslash-in-light-syntax }
7441 {
7442   You-can't-use~\token_to_str:N \\~to-separate-rows~because~you-have-used~
7443   the~key~'light-syntax'~.~You-must-use~the~character~'\l_@@_end_of_row_tl'~
7444   (set-by~the~key~'end-of-row')~.~This-error-is-fatal.
7445 }
7446 \@@_msg_new:nn { standard-cline-in-document }
7447 {
7448   The~key~'standard-cline'~is-available-only~in~the~preamble.\\
7449   If-you-go-on~this-command-will-be-ignored.
7450 }
7451 \@@_msg_new:nn { bad-value-for-baseline }
7452 {
7453   The~value-given~to~'baseline'~(\int_use:N \l_tmpa_int)~is-not~
7454   valid.~The~value~must~be~between~\int_use:N \l_@@_first_row_int\ and~
7455   \int_use:N \g_@@_row_total_int\ or~equal~to~'t',~'c'~or~'b'.\\
7456   If-you-go-on,~a~value~of~1~will~be~used.
7457 }
7458 \@@_msg_new:nn { Invalid-name-format }
7459 {
7460   You-can't-give-the-name~'\l_keys_value_tl'~to~a~\token_to_str:N
7461   \SubMatrix.\\
7462   A~name~must~be~accepted~by~the~regular-expression~[A-Za-z][A-Za-z0-9]*.\\
7463   If-you-go-on,~this-key-will-be-ignored.
7464 }
7465 \@@_msg_new:nn { Wrong-line-in-SubMatrix }
7466 {
7467   You-try-to-draw~a~#1~line~of~number~'#2'~in~a~
7468   \token_to_str:N \SubMatrix\ of~your~\@@_full_name_env:\ but~that~
7469   number~is~not~valid.~If-you-go-on,~it~will~be~ignored.
7470 }
7471 \@@_msg_new:nn { impossible-delimiter }
7472 {
7473   It's-impossible~to~draw~the~#1~delimiter~of~your~
7474   \token_to_str:N \SubMatrix\ because~all~the~cells~are~empty~
7475   in~that~column.
7476   \bool_if:NT \l_@@_submatrix_slim_bool

```

```

7477     { ~Maybe-you-should~try~without~the~key~'slim'. } \\
7478     If~you~go~on,~this~\token_to_str:N \SubMatrix\ will~be~ignored.
7479 }

7480 \@@_msg_new:nn { width~without~X~columns }
7481 {
7482     You~have~used~the~key~'width'~but~you~have~put~no~'X'~column. \\
7483     If~you~go~on,~that~key~will~be~ignored.
7484 }

7485 \@@_msg_new:nn { empty~environment }
7486 { Your~\@@_full_name_env:\ is~empty.~This~error~is~fatal. }

7487 \@@_msg_new:nn { No~letter~and~no~command }
7488 {
7489     Your~use~of~'custom~line'~is~no~op~since~you~don't~have~used~the~
7490     key~'letter'~(for~a~letter~for~vertical~rules)~nor~the~key~'command'~
7491     (to~draw~horizontal~rules).\\
7492     However,~you~can~go~on.
7493 }

7494 \@@_msg_new:nn { Forbidden~letter }
7495 {
7496     You~can't~use~the~letter~'\l_@@_letter_str'~for~a~customized~line.\\
7497     If~you~go~on,~it~will~be~ignored.
7498 }

7499 \@@_msg_new:nn { key~width~without~key~tikz }
7500 {
7501     In~'custom~line',~you~have~used~'width'~without~'tikz'.~That's~not~correct.~
7502     If~you~go~on,~that~key~'width'~will~be~discarded.
7503 }

7504 \@@_msg_new:nn { Several~letters }
7505 {
7506     You~must~use~only~one~letter~as~value~for~the~key~'letter'~(and~
7507     have~used~'\l_@@_letter_str').\\
7508     If~you~go~on,~it~will~be~ignored.
7509 }

7510 \@@_msg_new:nn { Delimiter~with~small }
7511 {
7512     You~can't~put~a~delimiter~in~the~preamble~of~your~\@@_full_name_env:\
7513     because~the~key~'small'~is~in~force.\\
7514     This~error~is~fatal.
7515 }

7516 \@@_msg_new:nn { unknown~cell~for~line~in~CodeAfter }
7517 {
7518     Your~command~\token_to_str:N\line\{#1\}\{#2\}~in~the~'code~after'~
7519     can't~be~executed~because~a~cell~doesn't~exist.\\
7520     If~you~go~on~this~command~will~be~ignored.
7521 }

7522 \@@_msg_new:nnn { Duplicate~name~for~SubMatrix }
7523 {
7524     The~name~'#1'~is~already~used~for~a~\token_to_str:N \SubMatrix\
7525     in~this~\@@_full_name_env:.\
7526     If~you~go~on,~this~key~will~be~ignored.\\
7527     For~a~list~of~the~names~already~used,~type~H~<return>.
7528 }
7529 {
7530     The~names~already~defined~in~this~\@@_full_name_env:\ are:~
7531     \seq_use:Nnnn \g_@@_submatrix_names_seq { ~and~ } { ,~ } { ~and~ }.
7532 }

7533 \@@_msg_new:nn { r~or~l~with~preamble }
7534 {
7535     You~can't~use~the~key~'\l_keys_key_str'~in~your~\@@_full_name_env:~

```

```

7536     You-must-specify-the-alignment-of-your-columns-with-the-preamble-of-
7537     your~\@@_full_name_env:.\
7538     If-you-go-on,~this-key-will-be-ignored.
7539 }

7540 \@@_msg_new:nn { Hdotsfor~in~col~0 }
7541 {
7542     You-can't-use~\token_to_str:N \Hdotsfor\ in-an-exterior~column-of~
7543     the~array.~This-error-is-fatal.
7544 }

7545 \@@_msg_new:nn { bad~corner }
7546 {
7547     #1-is-an~incorrect-specification~for~a~corner~(in~the~keys~
7548     'corners'~and~'except-corners').~The~available~
7549     values~are:~NW,~SW,~NE~and~SE.\\
7550     If-you-go-on,~this-specification-of~corner~will-be-ignored.
7551 }

7552 \@@_msg_new:nn { bad~border }
7553 {
7554     #1-is-an~incorrect-specification~for~a~border~(in~the~key~
7555     'borders'~of~the~command~\token_to_str:N \Block).~The~available~
7556     values~are:~left,~right,~top~and~bottom.\\
7557     If-you-go-on,~this-specification-of~border~will-be-ignored.
7558 }

7559 \@@_msg_new:nn { tikz~key~without~tikz }
7560 {
7561     You-can't-use~the~key~'tikz'~for~the~command~'\token_to_str:N
7562     \Block'~because~you~have~not~loaded~Tikz.~
7563     If-you-go-on,~this-key~will-be-ignored.
7564 }

7565 \@@_msg_new:nn { last~col~non~empty~for~NiceArray }
7566 {
7567     In~the~\@@_full_name_env:,~you~must~use~the~key~
7568     'last~col'~without~value.\\
7569     However,~you~can~go~on~for~this~time~
7570     (the~value~'\l_keys_value_tl'~will-be-ignored).
7571 }

7572 \@@_msg_new:nn { last~col~non~empty~for~NiceMatrixOptions }
7573 {
7574     In~\NiceMatrixoptions,~you~must~use~the~key~
7575     'last~col'~without~value.\\
7576     However,~you~can~go~on~for~this~time~
7577     (the~value~'\l_keys_value_tl'~will-be-ignored).
7578 }

7579 \@@_msg_new:nn { Block~too~large~1 }
7580 {
7581     You~try~to~draw~a~block~in~the~cell~#1-#2~of~your~matrix~but~the~matrix~is~
7582     too~small~for~that~block. \\
7583 }

7584 \@@_msg_new:nn { Block~too~large~2 }
7585 {
7586     The~preamble~of~your~\@@_full_name_env:\ announces~\int_use:N
7587     \g_@@_static_num_of_col_int\
7588     columns~but~you~use~only~\int_use:N \c@jCol\ and~that's~why~a~block~
7589     specified~in~the~cell~#1-#2~can't~be~drawn.~You~should~add~some~ampersands~
7590     (&)~at~the~end~of~the~first~row~of~your~
7591     \@@_full_name_env:.\
7592     If-you-go-on,~this~block~and~maybe~others~will-be-ignored.
7593 }

7594 \@@_msg_new:nn { unknown~column~type }
7595 {

```

```

7596 The~column~type~'#1'~in~your~\@@_full_name_env:\
7597 is~unknown. \\
7598 This~error~is~fatal.
7599 }

7600 \@@_msg_new:nn { colon~without~arydshln }
7601 {
7602   The~column~type~':~'~in~your~\@@_full_name_env:\
7603   is~unknown.~If~you~want~to~use~':~'~of~'arydshln',~you~should~
7604   load~that~package.~If~you~want~a~dotted~line~of~'nicematrix',~you~
7605   should~use~'\l_@@_letter_for_dotted_lines_str'.\\
7606   This~error~is~fatal.
7607 }

7608 \@@_msg_new:nn { tabularnote~forbidden }
7609 {
7610   You~can't~use~the~command~\token_to_str:N\tabularnote\
7611   ~in~a~\@@_full_name_env:~.~This~command~is~available~only~in~
7612   \{NiceTabular\},~\{NiceArray\}~and~\{NiceMatrix\}. \\
7613   If~you~go~on,~this~command~will~be~ignored.
7614 }

7615 \@@_msg_new:nn { borders~forbidden }
7616 {
7617   You~can't~use~the~key~'borders'~of~the~command~\token_to_str:N \Block\
7618   because~the~option~'rounded~corners'~
7619   is~in~force~with~a~non~zero~value.\\
7620   If~you~go~on,~this~key~will~be~ignored.
7621 }

7622 \@@_msg_new:nn { bottomrule~without~booktabs }
7623 {
7624   You~can't~use~the~key~'tabular/bottomrule'~because~you~haven't~
7625   loaded~'booktabs'.\\
7626   If~you~go~on,~this~key~will~be~ignored.
7627 }

7628 \@@_msg_new:nn { enumitem~not~loaded }
7629 {
7630   You~can't~use~the~command~\token_to_str:N\tabularnote\
7631   ~because~you~haven't~loaded~'enumitem'.\\
7632   If~you~go~on,~this~command~will~be~ignored.
7633 }

7634 \@@_msg_new:nn { tikz~in~custom~line~without~tikz }
7635 {
7636   You~have~used~the~key~'tikz'~in~the~definition~of~a~
7637   customized~line~(with~'custom~line')~but~Tikz~is~not~loaded.~
7638   You~can~go~on~but~you~will~have~another~error~if~you~actually~
7639   use~that~custom~line.
7640 }

7641 \@@_msg_new:nn { color~in~custom~line~with~tikz }
7642 {
7643   In~a~'custom~line',~you~have~used~both~'tikz'~and~'color',~
7644   which~is~forbidden~(you~should~use~'color'~inside~the~key~'tikz').~
7645   If~you~go~on,~the~key~'color'~will~be~discarded.
7646 }

7647 \@@_msg_new:nn { Wrong~last~row }
7648 {
7649   You~have~used~'last~row=\int_use:N \l_@@_last_row_int'~but~your~
7650   \@@_full_name_env:\ seems~to~have~\int_use:N \c@iRow \ rows.~
7651   If~you~go~on,~the~value~of~\int_use:N \c@iRow \ will~be~used~for~
7652   last~row.~You~can~avoid~this~problem~by~using~'last~row'~
7653   without~value~(more~compilations~might~be~necessary).
7654 }

7655 \@@_msg_new:nn { Yet~in~env }

```

```

7656 { Environments-of-nicematrix-can't-be-nested.\\ This-error-is-fatal. }
7657 \@@_msg_new:nn { Outside-math-mode }
7658 {
7659   The~\@@_full_name_env:\ can~be-used-only~in-math-mode~
7660   (and-not-in~\token_to_str:N \vcenter).\\
7661   This-error-is-fatal.
7662 }
7663 \@@_msg_new:nn { One-letter-allowed }
7664 {
7665   The-value-of-key~'\l_keys_key_str'~must-be-of-length-1.\\
7666   If-you-go-on,~it-will-be-ignored.
7667 }
7668 \@@_msg_new:nn { varwidth-not-loaded }
7669 {
7670   You-can't-use-the-column-type-'V'-because-'varwidth'~is-not~
7671   loaded.\\
7672   If-you-go-on,~your~column-will-behave-like-'p'.
7673 }
7674 \@@_msg_new:nnn { Unknown-key-for-Block }
7675 {
7676   The-key~'\l_keys_key_str'~is-unknown~for~the-command~\token_to_str:N
7677   \Block.\\ If-you-go-on,~it-will-be-ignored. \\
7678   For-a-list-of-the-available-keys,~type-H<return>.
7679 }
7680 {
7681   The-available-keys-are~(in~alphabetic-order):~b,~borders,~c,~draw,~fill,~
7682   hlines,~hvlines,~l,~line-width,~name,~rounded-corners,~r,~respect-arraystretch,
7683   ~t,~tikz~and~vlines.
7684 }
7685 \@@_msg_new:nn { Version-of-siunitx-too-old }
7686 {
7687   You-can't-use-'S'~columns-because-your-version-of-'siunitx'~
7688   is-too-old.~You-need-at-least-v3.0.\\
7689   This-error-is-fatal.
7690 }
7691 \@@_msg_new:nnn { Unknown-key-for-Brace }
7692 {
7693   The-key~'\l_keys_key_str'~is-unknown~for~the-commands~\token_to_str:N
7694   \UnderBrace\ and~\token_to_str:N \OverBrace.\\
7695   If-you-go-on,~it-will-be-ignored. \\
7696   For-a-list-of-the-available-keys,~type-H<return>.
7697 }
7698 {
7699   The-available-keys-are~(in~alphabetic-order):~left-shorten,~
7700   right-shorten,~shorten~(which-fixes-both-left-shorten-and~
7701   right-shorten)~and~yshift.
7702 }
7703 \@@_msg_new:nnn { Unknown-key-for-CodeAfter }
7704 {
7705   The-key~'\l_keys_key_str'~is-unknown.\\
7706   If-you-go-on,~it-will-be-ignored. \\
7707   For-a-list-of-the-available-keys-in~\token_to_str:N
7708   \CodeAfter,~type-H<return>.
7709 }
7710 {
7711   The-available-keys-are~(in~alphabetic-order):~
7712   delimiters/color,~
7713   rules~(with~the~subkeys~'color'~and~'width'),~
7714   sub-matrix~(several-subkeys)~
7715   and~xdots~(several-subkeys).~
7716   The-latter-is-for~the-command~\token_to_str:N \line.

```

```

7717     }
7718 \@@_msg_new:nnn { Unknown~key~for~SubMatrix }
7719 {
7720   The~key~'\l_keys_key_str'~is~unknown.\\
7721   If~you~go~on,~this~key~will~be~ignored. \\
7722   For~a~list~of~the~available~keys~in~\token_to_str:N
7723   \SubMatrix,~type~H~<return>.
7724 }
7725 {
7726   The~available~keys~are~(in~alphabetic~order):~
7727   'delimiters/color',~
7728   'extra-height',~
7729   'hlines',~
7730   'hvlines',~
7731   'left-xshift',~
7732   'name',~
7733   'right-xshift',~
7734   'rules'~(with~the~subkeys~'color'~and~'width'),~
7735   'slim',~
7736   'vlines'~and~'xshift'~(which~sets~both~'left-xshift'~
7737   and~'right-xshift').\\
7738 }
7739 \@@_msg_new:nnn { Unknown~key~for~notes }
7740 {
7741   The~key~'\l_keys_key_str'~is~unknown.\\
7742   If~you~go~on,~it~will~be~ignored. \\
7743   For~a~list~of~the~available~keys~about~notes,~type~H~<return>.
7744 }
7745 {
7746   The~available~keys~are~(in~alphabetic~order):~
7747   bottomrule,~
7748   code-after,~
7749   code-before,~
7750   enumitem-keys,~
7751   enumitem-keys-para,~
7752   para,~
7753   label-in-list,~
7754   label-in-tabular~and~
7755   style.
7756 }
7757 \@@_msg_new:nnn { Unknown~key~for~RowStyle }
7758 {
7759   The~key~'\l_keys_key_str'~is~unknown~for~the~command~
7760   \token_to_str:N \RowStyle. \\
7761   If~you~go~on,~it~will~be~ignored. \\
7762   For~a~list~of~the~available~keys,~type~H~<return>.
7763 }
7764 {
7765   The~available~keys~are~(in~alphabetic~order):~
7766   'bold',~
7767   'cell-space-top-limit',~
7768   'cell-space-bottom-limit',~
7769   'cell-space-limits',~
7770   'color',~
7771   'nb-rows'~and~
7772   'rowcolor'.
7773 }
7774 \@@_msg_new:nnn { Unknown~key~for~NiceMatrixOptions }
7775 {
7776   The~key~'\l_keys_key_str'~is~unknown~for~the~command~
7777   \token_to_str:N \NiceMatrixOptions. \\
7778   If~you~go~on,~it~will~be~ignored. \\

```

```

7779 For~a~list~of~the~*principal*~available~keys,~type~H~<return>.
7780 }
7781 {
7782   The~available~keys~are~(in~alphabetic~order):~
7783   allow~duplicate~names,~
7784   cell~space~bottom~limit,~
7785   cell~space~limits,~
7786   cell~space~top~limit,~
7787   code~for~first~col,~
7788   code~for~first~row,~
7789   code~for~last~col,~
7790   code~for~last~row,~
7791   corners,~
7792   custom~key,~
7793   create~extra~nodes,~
7794   create~medium~nodes,~
7795   create~large~nodes,~
7796   delimiters~(several~subkeys),~
7797   end~of~row,~
7798   first~col,~
7799   first~row,~
7800   hlines,~
7801   hvlines,~
7802   last~col,~
7803   last~row,~
7804   left~margin,~
7805   letter~for~dotted~lines,~
7806   light~syntax,~
7807   notes~(several~subkeys),~
7808   nullify~dots,~
7809   renew~dots,~
7810   renew~matrix,~
7811   respect~arraystretch,~
7812   right~margin,~
7813   rules~(with~the~subkeys~'color'~and~'width'),~
7814   small,~
7815   sub~matrix~(several~subkeys),
7816   vlines,~
7817   xdots~(several~subkeys).
7818 }
7819 \@@_msg_new:nnn { Unknown~key~for~NiceArray }
7820 {
7821   The~key~'\l_keys_key_str'~is~unknown~for~the~environment~
7822   \{NiceArray\}. \\\
7823   If~you~go~on,~it~will~be~ignored. \\\
7824   For~a~list~of~the~*principal*~available~keys,~type~H~<return>.
7825 }
7826 {
7827   The~available~keys~are~(in~alphabetic~order):~
7828   b,~
7829   baseline,~
7830   c,~
7831   cell~space~bottom~limit,~
7832   cell~space~limits,~
7833   cell~space~top~limit,~
7834   code~after,~
7835   code~for~first~col,~
7836   code~for~first~row,~
7837   code~for~last~col,~
7838   code~for~last~row,~
7839   colortbl~like,~
7840   columns~width,~
7841   corners,~

```



```

7842     create-extra-nodes,~
7843     create-medium-nodes,~
7844     create-large-nodes,~
7845     delimiters/color,~
7846     extra-left-margin,~
7847     extra-right-margin,~
7848     first-col,~
7849     first-row,~
7850     hlines,~
7851     hvlines,~
7852     last-col,~
7853     last-row,~
7854     left-margin,~
7855     light-syntax,~
7856     name,~
7857     notes/bottomrule,~
7858     notes/para,~
7859     nullify-dots,~
7860     renew-dots,~
7861     respect-arraystretch,~
7862     right-margin,~
7863     rules~(with~the~subkeys~'color'~and~'width'),~
7864     small,~
7865     t,~
7866     tabularnote,~
7867     vlines,~
7868     xdots/color,~
7869     xdots/shorten~and~
7870     xdots/line-style.
7871 }

```

This error message is used for the set of keys NiceMatrix/NiceMatrix and NiceMatrix/pNiceArray (but not by NiceMatrix/NiceArray because, for this set of keys, there is also the keys t, c and b).

```

7872 \@@_msg_new:nnn { Unknown-key-for-NiceMatrix }
7873 {
7874   The-key~'\l_keys_key_str'~is-unknown~for~the~
7875   \@@_full_name_env:. \\\
7876   If-you-go-on,~it~will~be~ignored. \\\
7877   For-a-list-of-the~*principal*~available-keys,~type-H~<return>.
7878 }
7879 {
7880   The~available~keys~are~(in~alphabetic~order):~
7881   b,~
7882   baseline,~
7883   c,~
7884   cell-space-bottom-limit,~
7885   cell-space-limits,~
7886   cell-space-top-limit,~
7887   code-after,~
7888   code-for-first-col,~
7889   code-for-first-row,~
7890   code-for-last-col,~
7891   code-for-last-row,~
7892   colortbl-like,~
7893   columns-width,~
7894   corners,~
7895   create-extra-nodes,~
7896   create-medium-nodes,~
7897   create-large-nodes,~
7898   delimiters~(several~subkeys),~
7899   extra-left-margin,~
7900   extra-right-margin,~
7901   first-col,~
7902   first-row,~

```

```

7903 hlines,~
7904 hvlines,~
7905 l,~
7906 last-col,~
7907 last-row,~
7908 left-margin,~
7909 light-syntax,~
7910 name,~
7911 nullify-dots,~
7912 r,~
7913 renew-dots,~
7914 respect-arraystretch,~
7915 right-margin,~
7916 rules~(with~the~subkeys~'color'~and~'width'),~
7917 small,~
7918 t,~
7919 vlines,~
7920 xdots/color,~
7921 xdots/shorten~and~
7922 xdots/line-style.
7923 }
7924 \@@_msg_new:nnn { Unknown~key~for~NiceTabular }
7925 {
7926   The~key~'\l_keys_key_str'~is~unknown~for~the~environment~
7927   \{NiceTabular\}. \\
7928   If~you~go~on,~it~will~be~ignored. \\
7929   For~a~list~of~the~*principal*~available~keys,~type~H~<return>.
7930 }
7931 {
7932   The~available~keys~are~(in~alphabetic~order):~
7933   b,~
7934   baseline,~
7935   c,~
7936   cell-space-bottom-limit,~
7937   cell-space-limits,~
7938   cell-space-top-limit,~
7939   code-after,~
7940   code-for-first-col,~
7941   code-for-first-row,~
7942   code-for-last-col,~
7943   code-for-last-row,~
7944   colortbl-like,~
7945   columns-width,~
7946   corners,~
7947   custom-line,~
7948   create-extra-nodes,~
7949   create-medium-nodes,~
7950   create-large-nodes,~
7951   extra-left-margin,~
7952   extra-right-margin,~
7953   first-col,~
7954   first-row,~
7955   hlines,~
7956   hvlines,~
7957   last-col,~
7958   last-row,~
7959   left-margin,~
7960   light-syntax,~
7961   name,~
7962   notes/bottomrule,~
7963   notes/para,~
7964   nullify-dots,~
7965   renew-dots,~

```

```

7966   respect-arraystretch,~
7967   right-margin,~
7968   rules~(with~the~subkeys~'color'~and~'width'),~
7969   t,~
7970   tabularnote,~
7971   vlines,~
7972   xdots/color,~
7973   xdots/shorten~and~
7974   xdots/line-style.
7975 }

7976 \@@_msg_new:nnn { Duplicate~name }
7977 {
7978   The~name~'\l_keys_value_tl'~is~already~used~and~you~shouldn't~use~
7979   the~same~environment~name~twice.~You~can~go~on,~but,~
7980   maybe,~you~will~have~incorrect~results~especially~
7981   if~you~use~'columns-width=auto'.~If~you~don't~want~to~see~this~
7982   message~again,~use~the~key~'allow-duplicate-names'~in~
7983   '\token_to_str:N \NiceMatrixOptions'.\\
7984   For~a~list~of~the~names~already~used,~type~H~<return>. \\
7985 }
7986 {
7987   The~names~already~defined~in~this~document~are:~
7988   \seq_use:Nnnn \g_@@_names_seq { ~and~ } { ,~ } { ~and~ }.
7989 }

7990 \@@_msg_new:nn { Option~auto~for~columns~width }
7991 {
7992   You~can't~give~the~value~'auto'~to~the~key~'columns-width'~here.~
7993   If~you~go~on,~the~key~will~be~ignored.
7994 }

```

19 History

The successive versions of the file `nicematrix.sty` provided by TeXLive are available on the SVN server of TeXLive:

<https://www.tug.org/svn/texlive/trunk/Master/texmf-dist/tex/latex/nicematrix/nicematrix.sty>

Changes between versions 1.0 and 1.1

The dotted lines are no longer drawn with Tikz nodes but with Tikz circles (for efficiency).
Modification of the code which is now twice faster.

Changes between versions 1.1 and 1.2

New environment `{NiceArray}` with column types L, C and R.

Changes between version 1.2 and 1.3

New environment `{pNiceArrayC}` and its variants.

Correction of a bug in the definition of `{BNiceMatrix}`, `{vNiceMatrix}` and `{VNiceMatrix}` (in fact, it was a typo).

Options are now available locally in `{pNiceMatrix}` and its variants.

The names of the options are changed. The old names were names in “camel style”.

Changes between version 1.3 and 1.4

The column types `w` and `W` can now be used in the environments `{NiceArray}`, `{pNiceArrayC}` and its variants with the same meaning as in the package `array`.

New option `columns-width` to fix the same width for all the columns of the array.

Changes between version 1.4 and 2.0

The versions 1.0 to 1.4 of `nicematrix` were focused on the continuous dotted lines whereas the version 2.0 of `nicematrix` provides different features to improve the typesetting of mathematical matrices.

Changes between version 2.0 and 2.1

New implementation of the environment `{pNiceArrayRC}`. With this new implementation, there is no restriction on the width of the columns.

The package `nicematrix` no longer loads `mathtools` but only `amsmath`.

Creation of “medium nodes” and “large nodes”.

Changes between version 2.1 and 2.1.1

Small corrections: for example, the option `code-for-first-row` is now available in the command `\NiceMatrixOptions`.

Following a discussion on TeX StackExchange⁷⁵, Tikz externalization is now deactivated in the environments of the package `nicematrix`.⁷⁶

Changes between version 2.1.2 and 2.1.3

When searching the end of a dotted line from a command like `\Cdots` issued in the “main matrix” (not in the exterior column), the cells in the exterior column are considered as outside the matrix. That means that it’s possible to do the following matrix with only a `\Cdots` command (and a single `\Vdots`).

$$\begin{pmatrix} & C_j & \\ 0 & \vdots & 0 \\ & a \dots & \\ 0 & & 0 \end{pmatrix} L_i$$

Changes between version 2.1.3 and 2.1.4

Replacement of some options `0 { }` in commands and environments defined with `xparse` by `! 0 { }` (because a recent version of `xparse` introduced the specifier `!` and modified the default behaviour of the last optional arguments).

See www.texdev.net/2018/04/21/xparse-optional-arguments-at-the-end

Changes between version 2.1.4 and 2.1.5

Compatibility with the classes `revtex4-1` and `revtex4-2`.

Option `allow-duplicate-names`.

⁷⁵cf. tex.stackexchange.com/questions/450841/tikz-externalize-and-nicematrix-package

⁷⁶Before this version, there was an error when using `nicematrix` with Tikz externalization. In any case, it’s not possible to externalize the Tikz elements constructed by `nicematrix` because they use the options `overlay` and `remember picture`.

Changes between version 2.1.5 and 2.2

Possibility to draw horizontal dotted lines to separate rows with the command `\hdottedline` (similar to the classical command `\hline` and the command `\hdashline` of `arydshln`).

Possibility to draw vertical dotted lines to separate columns with the specifier “:” in the preamble (similar to the classical specifier “|” and the specifier “:” of `arydshln`).

Changes between version 2.2 and 2.2.1

Improvement of the vertical dotted lines drawn by the specifier “:” in the preamble.

Modification of the position of the dotted lines drawn by `\hdottedline`.

Changes between version 2.2.1 and 2.3

Compatibility with the column type `S` of `siunitx`.

Option `hlines`.

Changes between version 2.3 and 3.0

Modification of `\Hdotsfor`. Now `\Hdotsfor` erases the `\vlines` (of “|”) as `\hdotsfor` does.

Composition of exterior rows and columns on the four sides of the matrix (and not only on two sides) with the options `first-row`, `last-row`, `first-col` and `last-col`.

Changes between version 3.0 and 3.1

Command `\Block` to draw block matrices.

Error message when the user gives an incorrect value for `last-row`.

A dotted line can no longer cross another dotted line (excepted the dotted lines drawn by `\cdottedline`, the symbol “:” (in the preamble of the array) and `\line` in `code-after`).

The starred versions of `\Cdots`, `\Ldots`, etc. are now deprecated because, with the new implementation, they become pointless. These starred versions are no longer documented.

The vertical rules in the matrices (drawn by “|”) are now compatible with the color fixed by `colortbl`.

Correction of a bug: it was not possible to use the colon “:” in the preamble of an array when `pdf \LaTeX` was used with `french-babel` (because `french-babel` activates the colon in the beginning of the document).

Changes between version 3.1 and 3.2 (and 3.2a)

Option `small`.

Changes between version 3.2 and 3.3

The options `first-row`, `last-row`, `first-col` and `last-col` are now available in the environments `{NiceMatrix}`, `{pNiceMatrix}`, `{bNiceMatrix}`, etc.

The option `columns-width=auto` doesn’t need any more a second compilation.

The options `renew-dots`, `renew-matrix` and `transparent` are now available as package options (as said in the documentation).

The previous version of `nicematrix` was incompatible with a recent version of `expl3` (released 2019/09/30). This version is compatible.

Changes between version 3.3 and 3.4

Following a discussion on TeX StackExchange⁷⁷, optimization of Tikz externalization is disabled in the environments of `nicematrix` when the class `standalone` or the package `standalone` is used.

Changes between version 3.4 and 3.5

Correction on a bug on the two previous versions where the `code-after` was not executed.

Changes between version 3.5 and 3.6

LaTeX counters `iRow` and `jCol` available in the cells of the array.

Addition of `\normalbaselines` before the construction of the array: in environments like `{align}` of `amsmath` the value of `\baselineskip` is changed and if the options `first-row` and `last-row` were used in an environment of `nicematrix`, the position of the delimiters was wrong.

A warning is written in the `.log` file if an obsolete environment is used.

There is no longer artificial errors `Duplicate~name` in the environments of `amsmath`.

Changes between version 3.6 and 3.7

The four “corners” of the matrix are correctly protected against the four codes: `code-for-first-col`, `code-for-last-col`, `code-for-first-row` and `code-for-last-row`.

New command `\pAutoNiceMatrix` and its variants (suggestion of Christophe Bal).

Changes between version 3.7 and 3.8

New programming for the command `\Block` when the block has only one row. With this programming, the vertical rules drawn by the specifier “|” at the end of the block is actually drawn. In previous versions, they were not because the block of one row was constructed with `\multicolumn`. An error is raised when an obsolete environment is used.

Changes between version 3.8 and 3.9

New commands `\NiceMatrixLastEnv` and `\OnlyMainNiceMatrix`.

New options `create-medium-nodes` and `create-large-nodes`.

Changes between version 3.9 and 3.10

New option `light-syntax` (and `end-of-row`).

New option `dotted-lines-margin` for fine tuning of the dotted lines.

Changes between versions 3.10 and 3.11

Correction of a bug linked to `first-row` and `last-row`.

⁷⁷cf. tex.stackexchange.com/questions/510841/nicematrix-and-tikz-external-optimize

Changes between versions 3.11 and 3.12

Command `\rotate` in the cells of the array.

Options `vlines`, `hlines` and `hvlines`.

Option `baseline` pour `{NiceArray}` (not for the other environments).

The name of the Tikz nodes created by the command `\Block` has changed: when the command has been issued in the cell $i-j$, the name is $i-j$ -block and, if the creation of the “medium nodes” is required, a node $i-j$ -block-medium is created.

If the user tries to use more columns than allowed by its environment, an error is raised by `nicematrix` (instead of a low-level error).

The package must be loaded with the option `obsolete-environments` if we want to use the deprecated environments.

Changes between versions 3.12 and 3.13

The behaviour of the command `\rotate` is improved when used in the “last row”.

The option `dotted-lines-margin` has been renamed in `xdots/shorten` and the options `xdots/color` and `xdots/line-style` have been added for a complete customisation of the dotted lines.

In the environments without preamble (`{NiceMatrix}`, `{pNiceMatrix}`, etc.), it’s possible to use the options `l` (=L) or `r` (=R) to specify the type of the columns.

The starred versions of the commands `\Cdots`, `\Ldots`, `\Vdots`, `\Ddots` and `\Iddots` are deprecated since the version 3.1 of `nicematrix`. Now, one should load `nicematrix` with the option `starred-commands` to avoid an error at the compilation.

The code of `nicematrix` no longer uses Tikz but only PGF. By default, Tikz is *not* loaded by `nicematrix`.

Changes between versions 3.13 and 3.14

Correction of a bug (question 60761504 on [stackoverflow](#)).

Better error messages when the user uses `&` or `\\` when `light-syntax` is in force.

Changes between versions 3.14 and 3.15

It’s possible to put labels on the dotted lines drawn by `\Ldots`, `\Cdots`, `\Vdots`, `\Ddots`, `\Iddots`, `\Hdotsfor` and the command `\line` in the `code-after` with the tokens `_` and `^`.

The option `baseline` is now available in all the environments of `nicematrix`. Before, it was available only in `{NiceArray}`.

New keyword `\CodeAfter` (in the environments of `nicematrix`).

Changes between versions 3.15 and 4.0

New environment `{NiceTabular}`

Commands to color cells, rows and columns with a perfect result in the PDF.

Changes between versions 4.0 and 4.1

New keys `cell-space-top-limit` and `cell-space-bottom-limit`

New command `\diagbox`

The key `hvline` don’t draw rules in the blocks (commands `\Block`) and in the virtual blocks corresponding to the dotted lines.

Changes between versions 4.1 and 4.2

It’s now possible to write `\begin{pNiceMatrix}a&b\\c&d\end{pNiceMatrix}`² with the expected result.

Changes between versions 4.2 and 4.3

The horizontal centering of the content of a `\Block` is correct even when an instruction such as `!\qqquad` is used in the preamble of the array.

It's now possible to use the command `\Block` in the “last row”.

Changes between versions 4.3 and 4.4

New key `hvlines-except-corners`.

Changes between versions 4.4 and 5.0

Use of the standard column types `l`, `c` and `r` instead of `L`, `C` and `R`.

It's now possible to use the command `\diagbox` in a `\Block`.

Command `\tabularnote`

Changes between versions 5.0 and 5.1

The vertical rules specified by `|` in the preamble are not broken by `\hline\hline` (and other).

Environment `{NiceTabular*}`

Command `\Vdotsfor` similar to `\Hdotsfor`

The variable `\g_nicematrix_code_after_tl` is now public.

Changes between versions 5.1 and 5.2

The vertical rules specified by `|` or `||` in the preamble respect the blocks.

Key `respect-blocks` for `\rowcolors` (with a *s*) in the `code-before`.

The variable `\g_nicematrix_code_before_tl` is now public.

The key `baseline` may take in as value an expression of the form *line-i* to align the `\hline` in the row *i*.

The key `hvlines-except-corners` may take in as value a list of corners (eg: NW,SE).

Changes between versions 5.2 and 5.3

Keys `c`, `r` and `l` for the command `\Block`.

It's possible to use the key `draw-first` with `\Ddots` and `\Iddots` to specify which dotted line will be drawn first (the other lines will be drawn parallel to that one if parallelization is activated).

Changes between versions 5.3 and 5.4

Key `tabularnote`.

Different behaviour for the mono-column blocks.

Changes between versions 5.4 and 5.5

The user must never put `\omit` before `\CodeAfter`.

Correction of a bug: the tabular notes `\tabularnotes` were not composed when present in a block (except a mono-column block).

Changes between versions 5.5 and 5.6

Different behaviour for the mono-row blocks.

New command `\NotEmpty`.

Changes between versions 5.6 and 5.7

New key `delimiters-color`

Keys `fill`, `draw` and `line-width` for the command `\Block`.

Changes between versions 5.7 and 5.8

Keys `cols` and `restart` of the command `\rowcolors` in the `code-before`.

Modification of the behaviour of `\\` in the columns of type `p`, `m` or `b` (for a behaviour similar to the environments of `array`).

Better error messages for the command `\Block`.

Changes between versions 5.8 and 5.9

Correction of a bug: in the previous versions, it was not possible to use the key `line-style` for the continuous dotted lines when the Tikz library `babel` was loaded.

New key `cell-space-limits`.

Changes between versions 5.9 and 5.10

New command `\SubMatrix` available in the `\CodeAfter`.

It's possible to provide options (between brackets) to the keyword `\CodeAfter`.

A (non fatal) error is raised when the key `transparent`, which is deprecated, is used.

Changes between versions 5.10 and 5.11

It's now possible, in the `code-before` and in the `\CodeAfter`, to use the syntax `|(i-|j)` for the Tikz node at the intersection of the (potential) horizontal rule number i and the (potential) vertical rule number j .

Changes between versions 5.11 and 5.12

Keywords `\CodeBefore` and `\Body` (alternative syntax to the key `code-before`).

New key `delimiters/max-width`.

New keys `hlines`, `vlines` and `hvlines` for the command `\SubMatrix` in the `\CodeAfter`.

New key `rounded-corners` for the command `\Block`.

Changes between versions 5.12 and 5.13

New command `\arraycolor` in the `\CodeBefore` (with its key `except-corners`).

New key `borders` for the command `\Block`.

New command `\Hline` (for horizontal rules not drawn in the blocks).

The keys `vlines` and `hlines` takes in as value a (comma-separated) list of numbers (for the rules to draw).

Changes between versions 5.13 and 5.14

Nodes of the form (1.5) , (2.5) , (3.5) , etc.

Keys `t` and `b` for the command `\Block`.

Key `corners`.

Changes between versions 5.14 and 5.15

Key `hvlines` for the command `\Block`.

The commands provided by `nicematrix` to color cells, rows and columns don't color the cells which are in the “corners” (when the key `corner` is used).

It's now possible to specify delimiters for submatrices in the preamble of an environment.

The version 5.15b is compatible with the version 3.0+ of `siunitx` (previous versions were not).

Changes between versions 5.15 and 5.16

It's now possible to use the cells corresponding to the contents of the nodes (of the form `i-j`) in the `\CodeBefore` when the key `create-cell-nodes` of that `\CodeBefore` is used. The medium and the large nodes are also available if the corresponding keys are used.

Changes between versions 5.16 and 5.17

The key `define-L-C-R` (only available at load-time) now raises a (non fatal) error.

Keys `L`, `C` and `R` for the command `\Block`.

Key `hvlines-except-borders`.

It's now possible to use a key `l`, `r` or `c` with the command `\pAutoNiceMatrix` (and the similar ones).

Changes between versions 5.17 and 5.18

New command `\RowStyle`

Changes between versions 5.18 and 5.19

New key `tikz` for the command `\Block`.

Changes between versions 5.19 and 6.0

Columns `X` and environment `{NiceTabularX}`.

Command `\rowlistcolors` available in the `\CodeBefore`.

In columns with fixed width, the blocks are composed as paragraphs (wrapping of the lines).

The key `define-L-C-R` has been deleted.

Changes between versions 6.0 and 6.1

Better computation of the widths of the `X` columns.

Key `\color` for the command `\RowStyle`.

Changes between versions 6.1 and 6.2

Better compatibility with the classes `revtex4-1` and `revtex4-2`.

Key `vlines-in-sub-matrix`.

Changes between versions 6.2 and 6.3

Keys `nb-rows`, `rowcolor` and `bold` for the command `\RowStyle`

Key `name` for the command `\Block`.

Support for the columns `V` of `varwidth`.

Changes between versions 6.3 and 6.4

New commands `\UnderBrace` and `\OverBrace` in the `\CodeAfter`.

Correction of a bug of the key `baseline` (cf. question 623258 on TeX StackExchange).

Correction of a bug with the columns `V` of `varwidth`.

Correction of a bug: the use of `\hdottedline` and `:` in the preamble of the array (of another letter specified by `letter-for-dotted-lines`) was incompatible with the key `xdots/line-style`.

Changes between versions 6.4 and 6.5

Key `custom-line` in `\NiceMatrixOptions`.

Key `respect-arraystretch`.

Changes between version 6.5 and 6.6

Keys `tikz` and `width` in `custom-line`.

Index

The italic numbers denote the pages where the corresponding entry is described, numbers underlined point to the definition, all others indicate the places where it is used.

Symbols		
<code>@@</code> commands:		
<code>\@@_Block:</code>	1229, 5717	
<code>\@@_Block_i</code>	5722, 5723, 5727	
<code>\@@_Block_ii:nnnnn</code>	5727, 5728	
<code>\@@_Block_iv:nnnnn</code>	5765, 5769	
<code>\@@_Block_iv:nnnnnn</code>	5995, 5997	
<code>\@@_Block_v:nnnnn</code>	5766, 5892	
<code>\@@_Block_v:nnnnnn</code>	6024, 6027	
<code>\@@_Cdots</code>	1149, 1220, 3993	
<code>\g_@@_Cdots_lines_tl</code>	1250, 3185	
<code>\@@_CodeAfter:</code>	1233, 6639	
<code>\@@_CodeAfter_i:</code>	883, 2832, 2877, 6640	
<code>\@@_CodeAfter_ii:n</code>	6639, 6640, 6641, 6651	
<code>\@@_CodeAfter_iv:n</code>	6644, 6646	
<code>\@@_CodeAfter_keys:</code>	3126, 3145	
<code>\@@_CodeBefore:w</code>	1392, 1394	
<code>\@@_CodeBefore_keys:</code>	1372, 1389	
<code>\@@_Ddots</code>	1151, 1222, 4025	
<code>\g_@@_Ddots_lines_tl</code>	1253, 3183	
<code>\g_@@_HVdotsfor_lines_tl</code>	1255, 3181, 4109, 4186, 7340	
<code>\@@_Hdotsfor:</code>	1154, 1227, 4085	
<code>\@@_Hdotsfor:nnnn</code>	4111, 4124	
<code>\@@_Hdotsfor_i</code>	4094, 4100, 4107	
<code>\@@_Hline:</code>	1225, 5161	
<code>\@@_Hline_i:n</code>	5161, 5162, 5168	
<code>\@@_Hline_ii:nn</code>	5165, 5168	
<code>\@@_Hline_iii:n</code>	5166, 5169	
<code>\@@_Hspace:</code>	1226, 4079	
<code>\@@_Iddots</code>	1152, 1223, 4049	
<code>\g_@@_Iddots_lines_tl</code>	1254, 3184	
<code>\@@_Ldots</code>	1148, 1153, 1219, 3977	
<code>\g_@@_Ldots_lines_tl</code>	1251, 3186	
<code>\l_@@_Matrix_bool</code>	245, 1667, 1691, 2436, 2776, 2780, 2788, 2945	
<code>\l_@@_NiceArray_bool</code>	242, 402, 1296, 1535, 1607, 1730, 1737, 1749, 2436, 2764, 2776, 2780, 2788, 2921, 4952, 4956, 5107, 5117, 5147, 5151, 6543	
<code>\g_@@_NiceMatrixBlock_int</code>	227, 5475, 5480, 5483, 5494	
<code>\l_@@_NiceTabular_bool</code>	184, 243, 888, 1076, 1371, 1374, 1502, 1640, 1644, 1738, 1750, 2765, 2976, 2997, 3006, 5799, 5901, 6551	
<code>\@@_NotEmpty:</code>	1235, 2968	
<code>\@@_OnlyMainNiceMatrix:n</code>	1231, 4739	
<code>\@@_OnlyMainNiceMatrix_i:n</code>	4742, 4749, 4752	
<code>\@@_OverBrace</code>	3120, 7102	
<code>\@@_RowStyle:n</code>	1236, 4337	
<code>\@@_S:</code>	215, 1780	
<code>\@@_SubMatrix</code>	3118, 6792	
<code>\@@_SubMatrix_in_code_before</code>	1364, 6772	
<code>\@@_UnderBrace</code>	3119, 7097	
<code>\@@_V:</code>	1695, 1776	
<code>\@@_Vdots</code>	1150, 1221, 4009	
<code>\g_@@_Vdots_lines_tl</code>	1252, 3182	
<code>\@@_Vdotsfor:</code>	1228, 4184	
<code>\@@_Vdotsfor:nnnn</code>	4188, 4199	
<code>\@@_W:</code>	1694, 1779, 2293	
<code>\@@_X</code>	1788, 2992	
<code>\l_@@_X_column_bool</code>	247, 2185, 5763	
<code>\l_@@_X_columns_aux_bool</code>	274, 1560, 2174	
<code>\l_@@_X_columns_dim</code>	275, 1561, 1570, 1576, 2177	
<code>\@@_actually_color:</code>	1373, 4424	
<code>\@@_actually_diagbox:nnnnnn</code>	5776, 6108, 6563, 6580	
<code>\@@_actually_draw_Cdots:</code>	3551, 3555	
<code>\@@_actually_draw_Ddots:</code>	3701, 3705	
<code>\@@_actually_draw_Iddots:</code>	3749, 3753	
<code>\@@_actually_draw_Ldots:</code>	3512, 3516, 4175	
<code>\@@_actually_draw_Vdots:</code>	3633, 3637, 4250	

```

\@@_add_to_colors_seq:nn .....
    4410, 4422, 4423, 4447, 4456, 4465, 4474, 4578
\@@_adjust_pos_of_blocks_seq: .. 3102, 3147
\@@_adjust_pos_of_blocks_seq_i:nnnnn .
    ..... 3150, 3152
\@@_adjust_size_box: .....
    ..... 961, 988, 2035, 2353, 2856, 2901
\@@_adjust_to_submatrix:nn .....
    ..... 3396, 3499, 3538, 3619, 3694, 3742
\@@_adjust_to_submatrix:nnnnnn . 3403, 3405
\@@_after_array: ..... 1676, 3010
\g_@@_after_col_zero_bool .....
    ..... 276, 1116, 2833, 4091
\@@_analyze_end:Nn ..... 2584, 2629
\l_@@_argspec_tl 3975, 3976, 3977, 3993,
    4009, 4025, 4049, 4105, 4106, 4107, 4182,
    4183, 4184, 4260, 4261, 4262, 6790, 6791, 6792
\@@_array: ..... 1074, 2585, 2612
\@@_arraycolor ..... 1361, 4509
\c_@@_arydshln_loaded_bool ... 25, 37, 1805
\l_@@_auto_columns_width_bool .....
    ..... 510, 658, 2696, 2700, 5470
\g_@@_aux_found_bool .....
    ..... 1267, 1272, 1396, 1523, 1526
\g_@@_aux_tl ..... 248,
    1529, 1558, 1683, 3019, 3033, 3041, 3134, 5357
\l_@@_bar_at_end_of_pream_bool .....
    ..... 1731, 1874, 2768
\l_@@_baseline_tl 499, 500, 651, 652, 653,
    654, 1083, 1609, 2389, 2401, 2406, 2408,
    2413, 2418, 2508, 2509, 2513, 2518, 2520, 2525
\@@_begin_of_NiceMatrix:nn .... 2943, 2964
\@@_begin_of_row: .... 886, 911, 1179, 2834
\l_@@_block_auto_columns_width_bool ..
    ..... 1515, 2701, 5463, 5468, 5478, 5488
\g_@@_block_box_int ..... 321, 1496,
    5771, 5785, 5787, 5845, 5857, 5869, 5878, 5888
\l_@@_block_name_str .....
    ..... 5985, 6055, 6132, 6135, 6140
\@@_block_tikz:nnnnn ..... 6096, 6453
\g_@@_blocks_dp_dim .....
    ..... 238, 969, 972, 973, 5872, 5875
\g_@@_blocks_ht_dim .....
    ..... 237, 975, 978, 979, 5863, 5866
\g_@@_blocks_seq .....
    ..... 291, 1517, 2446, 5882, 5894, 5995
\g_@@_blocks_wd_dim .....
    ..... 236, 963, 966, 967, 5851, 5854
\c_@@_booktabs_loaded_bool 26, 40, 1168, 2479
\l_@@_borders_clist ..... 309, 5957,
    6069, 6395, 6411, 6413, 6415, 6417, 6436, 6448
\@@_brace:nnnnn ..... 7100, 7105, 7119
\l_@@_brace_left_shorten_bool .....
    ..... 7109, 7136, 7151
\l_@@_brace_right_shorten_bool .....
    ..... 7111, 7157, 7172
\l_@@_brace_yshift_dim ... 7114, 7193, 7225
\@@_cartesian_color:nn 4437, 4449, 4458, 4580
\@@_cartesian_path: ..... 4441, 4679
\@@_cartesian_path:n ..... 4489, 4621, 4679
\@@_cell_begin:w ..... 880, 1825,
    1955, 2026, 2057, 2184, 2302, 2323, 2344, 2954
\l_@@_cell_box ..... 887, 935, 937, 943,
    949, 952, 956, 965, 966, 971, 972, 977, 978,
    989, 990, 991, 992, 994, 997, 1001, 1003,
    1022, 1037, 1039, 1046, 1047, 1060, 1170,
    1280, 1282, 1983, 1987, 1991, 1994, 2025,
    2036, 2187, 2343, 2354, 2835, 2859, 2862,
    2864, 2881, 2904, 2908, 6116, 6236, 6273, 6558
\@@_cell_end: ..... 982, 1827,
    1974, 2031, 2062, 2193, 2304, 2333, 2349, 2954
\l_@@_cell_space_bottom_limit_dim ...
    ..... 485, 563, 992, 4383
\l_@@_cell_space_top_limit_dim .....
    ..... 484, 561, 990, 4372
\@@_cellcolor .. 1355, 4491, 4503, 4504, 4710
\@@_cellcolor_tabular ..... 1158, 4704
\g_@@_cells_seq ..... 2623, 2624, 2625, 2627
\@@_center_cell_box: ..... 1933, 1978
\@@_chessboardcolors ..... 1363, 4496
\@@_cline ..... 155, 1218
\@@_cline_i:nn ..... 156, 157, 177, 180
\@@_cline_i:w ..... 157, 158
\@@_cline_ii:w ..... 162, 164
\@@_cline_iii:w ..... 161, 165, 166
\l_@@_code_before_bool .....
    .. 280, 648, 675, 1090, 1287, 1318, 1532,
    2643, 2660, 2678, 2709, 2735, 2771, 2808, 3139
\g_@@_code_before_tl . 1521, 1530, 1533, 3136
\l_@@_code_before_tl .....
    ..... 279, 647, 1317, 1372, 1533
\l_@@_code_for_first_col_tl ..... 580, 2846
\l_@@_code_for_first_row_tl . 584, 899, 6205
\l_@@_code_for_last_col_tl ..... 582, 2890
\l_@@_code_for_last_row_tl . 586, 906, 6208
\l_@@_code_tl ..... 268, 6767, 7003
\l_@@_col_max_int .....
    ..... 303, 3262, 3273, 3341, 3401, 3418
\l_@@_col_min_int .....
    ..... 302, 3267, 3330, 3335, 3399, 3416
\g_@@_col_total_int .....
    ..... 232, 1005, 1246, 1326, 1337,
    1409, 1593, 2274, 2275, 2728, 2729, 2758,
    2811, 2815, 2820, 2821, 2880, 3014, 3016,
    3028, 3586, 3604, 3664, 4245, 4729, 5521,
    5531, 5565, 5652, 6007, 6243, 6828, 6877, 7125
\l_@@_col_width_dim .....
    ..... 229, 230, 1954, 2024, 5804, 5809
\@@_color_index:n ..... 4568, 4589, 4592
\l_@@_color_int .....
    ..... 4532, 4533, 4550, 4551, 4571, 4582
\l_@@_color_tl .....
    311, 4565, 4566, 4576, 4579, 5712, 5789, 5791
\g_@@_colors_seq 1368, 4413, 4417, 4418, 4428
\l_@@_colors_seq 4527, 4528, 4572, 4591, 4593
\@@_colortbl_like: ..... 1156, 1237
\l_@@_colortbl_like_bool 482, 674, 1237, 1719
\c_@@_colortbl_loaded_bool . 109, 113, 1187
\l_@@_cols_tl .....
    4440, 4488, 4519, 4529, 4530, 4580, 4627, 4630
\g_@@_cols_vlism_seq .....
    ..... 256, 1285, 1714, 1797, 6905
\@@_columncolor ..... 1362, 4452
\@@_columncolor_preamble ..... 1160, 4727
\c_@@_columncolor_regex ..... 68, 1722

```

<code>\l_@@_columns_width_dim</code>	<code>\g_@@_dp_row_zero_dim</code>
..... 228, 659, 796, 2697, 2703, 5476, 5482 934, 935, 1197, 1198, 1619, 2502, 2541
<code>\g_@@_com_or_env_str</code>	<code>\@@_draw_Cdots:nnn</code>
260, 263	3536
<code>\l_@@_command_str</code> 5189, 5210, 5230, 5251, 5265	<code>\@@_draw_Ddots:nnn</code>
<code>\@@_computations_for_large_nodes:</code> ...	3692
..... 5592, 5605, 5610	<code>\@@_draw_Iddots:nnn</code>
<code>\@@_computations_for_medium_nodes:</code> ..	3740
..... 5512, 5581, 5591, 5602	<code>\@@_draw_Ldots:nnn</code>
<code>\@@_compute_a_corner:nnnnnn</code>	3497
..... 5345, 5347, 5349, 5351, 5364	<code>\@@_draw_Vdots:nnn</code>
<code>\@@_compute_corners:</code>	3617
3101, 5337	<code>\@@_draw_blocks:</code>
<code>\@@_compute_i_j:nn</code>	2446, 5992
6801, 6825, 7122	<code>\@@_draw_dotted_lines:</code>
<code>\@@_construct_preamble:</code>	3100, 3170
1293, 1688	<code>\@@_draw_dotted_lines_i:</code>
<code>\l_@@_corners_cells_seq</code>	3173, 3177
..... 295, 4624, 4664, 4835, 4841, 4848,	<code>\l_@@_draw_first_bool</code> .
5016, 5022, 5029, 5339, 5355, 5359, 5360, 5420	318, 4040, 4064, 4075
<code>\l_@@_corners_clist</code>	<code>\@@_draw_hlines:</code>
..... 504, 636, 641, 4809, 4990, 5340	3103, 5143
<code>\@@_create_blocks_nodes:</code>	<code>\@@_draw_line:</code>
1346, 1441	3534,
<code>\@@_create_col_nodes:</code>	3579, 3690, 3738, 3786, 3788, 4309, 4928, 5123
2588, 2616, 2635	<code>\@@_draw_line_ii:nn</code>
<code>\@@_create_diag_nodes:</code> ...	4289, 4293
1343, 3049, 3203	<code>\@@_draw_line_iii:nn</code>
<code>\@@_create_extra_nodes:</code> ..	4296, 4300
1439, 2445, 5502	<code>\@@_draw_standard_dotted_line:</code> .
<code>\@@_create_large_nodes:</code>	3795, 3826
5510, 5586	<code>\@@_draw_standard_dotted_line_i:</code> 3889, 3893
<code>\@@_create_medium_and_large_nodes:</code> ..	<code>\l_@@_draw_tl</code>
..... 5507, 5597	307, 5951,
<code>\@@_create_medium_nodes:</code>	5955, 6057, 6288, 6294, 6296, 6298, 6335, 6336
5508, 5576	<code>\@@_draw_unstandard_dotted_line:</code> 3796, 3798
<code>\@@_create_nodes:</code> 5583, 5594, 5604, 5607, 5648	<code>\@@_draw_unstandard_dotted_line:n</code> ...
<code>\@@_create_one_block_node:nnnnn</code> 1447, 1450 3801, 3804
<code>\@@_create_row_node:</code>	<code>\@@_draw_unstandard_dotted_line:nnn</code> ..
1086, 1119, 1169 3806, 3811, 3825
<code>\@@_custom_line:n</code>	<code>\@@_draw_vlines:</code>
554, 5187	3104, 4948
<code>\@@_custom_line_i:n</code>	<code>\g_@@_empty_cell_bool</code> ...
5224, 5275	288, 996, 1006,
<code>\@@_cut_on_hyphen:w</code>	2869, 2916, 3991, 4007, 4023, 4047, 4070, 4081
341,	<code>\l_@@_end_int</code>
354, 4481, 4486, 4546, 4634, 4635, 4657,	4763,
4658, 4688, 4689, 6306, 6315, 6346, 6349,	4787, 4788, 4799, 4826, 4969, 4970, 4980, 5007
6369, 6372, 6400, 6403, 6776, 6781, 6807, 6814	<code>\l_@@_end_of_row_tl</code>
<code>\g_@@_ddots_int</code> 520, 521, 574, 2608, 2609, 7443
3081, 3721, 3722	<code>\c_@@_endpgfortikzpicture_tl</code>
<code>\@@_def_env:nnn</code> 52, 56, 3174, 4297
..... 2927, 2938, 2939, 2940, 2941, 2942	<code>\c_@@_enumitem_loaded_bool</code>
<code>\@@_define_com:nnn</code> 27, 43, 375, 702, 707, 718, 723
..... 6527, 6535, 6536, 6537, 6538, 6539	<code>\@@_env:</code>
<code>\@@_define_h_custom_line:nn</code>	222, 226, 920, 926,
5240, 5263	1023, 1029, 1043, 1051, 1095, 1101, 1107,
<code>\@@_delimiter:nnn</code>	1182, 1327, 1328, 1333, 1334, 1339, 1340,
... 2090, 2111, 2119, 2132, 2138, 2144, 6654	1352, 1406, 1407, 1412, 1415, 1418, 1421,
<code>\l_@@_delimiters_color_tl</code>	1430, 1431, 1436, 1437, 1463, 2644, 2647,
523,	2649, 2665, 2671, 2674, 2683, 2689, 2692,
734, 1385, 1632, 1633, 1650, 1651, 6633,	2714, 2720, 2723, 2740, 2746, 2752, 2778,
6692, 6693, 6720, 6742, 7055, 7056, 7079, 7080	2786, 2800, 2811, 2815, 2821, 3054, 3055,
<code>\l_@@_delimiters_max_width_bool</code>	3060, 3061, 3069, 3075, 3113, 3220, 3222,
..... 524, 732, 1655	3229, 3231, 3232, 3237, 3240, 3302, 3370,
<code>\g_@@_delta_x_one_dim</code>	3435, 3446, 3459, 3462, 3481, 3484, 3589,
3083, 3724, 3734	3592, 3607, 3610, 4138, 4156, 4213, 4231,
<code>\g_@@_delta_x_two_dim</code>	4282, 4284, 4303, 4306, 5375, 5394, 5412,
3085, 3772, 3782	5534, 5536, 5544, 5655, 5664, 5682, 6130,
<code>\g_@@_delta_y_one_dim</code>	6135, 6136, 6141, 6150, 6154, 6168, 6173,
3084, 3726, 3734	6185, 6191, 6192, 6195, 6213, 6250, 6669,
<code>\g_@@_delta_y_two_dim</code>	6672, 6844, 6846, 6851, 6853, 6880, 6882,
3086, 3774, 3782	6887, 6889, 6987, 6999, 7142, 7144, 7163, 7165
<code>\@@_diagbox:nn</code>	<code>\g_@@_env_int</code>
1234, 6559	. 221, 222, 224, 1514, 1524, 1527, 1682, 5691
<code>\@@_dotfill:</code>	<code>\@@_error:n</code>
6548	12, 378, 403, 533, 543,
<code>\@@_dotfill_i:</code>	596, 728, 786, 795, 804, 812, 830, 837, 845,
6553, 6555	846, 847, 853, 858, 859, 860, 874, 876, 877,
<code>\@@_dotfill_ii:</code>	878, 1387, 1554, 1588, 1598, 1670, 2015,
6552, 6555, 6556	2423, 2484, 2530, 4335, 4522, 5199, 5201,
<code>\@@_dotfill_iii:</code>	5206, 5211, 5219, 5228, 5947, 5990, 6393,
6556, 6557	6637, 6752, 6763, 6770, 7117, 7275, 7302, 7312
<code>\l_@@_dotted_bool</code>	
..... 329, 3794, 4771, 4857, 4859, 5038, 5040	
<code>\@@_double_int_eval:n</code>	
4256, 4270, 4271	
<code>\g_@@_dp_ante_last_row_dim</code>	
914, 1203	
<code>\g_@@_dp_last_row_dim</code>	
..... 914, 915, 1206, 1207, 1281, 1282, 1626	

```

\@@_error:nn ..... 7486, 7512, 7525, 7530, 7535, 7537, 7567,
..... 13, 666, 2093, 2139, 2145, 2169, 7586, 7591, 7596, 7602, 7611, 7650, 7659, 7875
..... 3980, 3983, 3996, 3999, 4012, 4015, 4029,
..... 4030, 4035, 4036, 4053, 4054, 4059, 4060,
..... 5353, 6398, 6757, 6829, 6859, 6862, 7128, 7129
\@@_error:nnn ..... 14, 4287, 6934, 6969
\@@_error_too_much_cols: ..... 1759, 7325
\@@_everycr: ..... 1112, 1192, 1195
\@@_everycr_i: ..... 1112, 1113
\l_@@_except_borders_bool .....
..... 515, 608, 4952, 4956, 5147, 5151
\@@_exec_code_before: ..... 1287, 1366
\@@_expand_clist:N ..... 346, 1166, 1167
\@@_expand_clist:NN ..... 4627, 4628, 4680
\l_@@_exterior_arraycolsep_bool .....
..... 501, 792, 1740, 1752, 2767
\l_@@_extra_left_margin_dim .....
..... 518, 624, 1309, 2867
\l_@@_extra_right_margin_dim .....
..... 519, 625, 1549, 2912, 3667
\@@_extract_brackets ..... 6081, 6278
\@@_extract_coords_values: ..... 5673, 5680
\@@_fatal:n ..... 15,
..... 252, 1505, 1810, 2067, 2071, 2100, 2593,
..... 2597, 2599, 2632, 4096, 7271, 7330, 7333, 7336
\@@_fatal:nn ..... 16, 1817, 2296
\l_@@_fill_tl ..... 306, 5949, 6079, 6081
\l_@@_final_i_int .....
..... 3090, 3249, 3254, 3257, 3282,
..... 3290, 3294, 3303, 3311, 3391, 3447, 3528,
..... 3601, 3607, 3610, 4129, 4157, 4225, 4235, 4237
\l_@@_final_j_int .....
..... 3091, 3250, 3255, 3262, 3267, 3273, 3283,
..... 3291, 3295, 3304, 3312, 3390, 3392, 3448,
..... 3481, 3484, 3492, 4150, 4160, 4162, 4204, 4233
\l_@@_final_open_bool .....
..... 3093, 3256, 3260, 3263, 3270, 3276, 3280,
..... 3296, 3525, 3560, 3565, 3576, 3640, 3650,
..... 3655, 3676, 3713, 3761, 3897, 3912, 3943,
..... 3944, 4127, 4151, 4163, 4202, 4226, 4238, 4279
\@@_find_extremities_of_line:nnnn ...
..... 3244, 3502, 3541, 3622, 3697, 3745
\l_@@_first_col_int .....
..... 143, 156, 332, 333, 576, 851,
..... 886, 1337, 1409, 1602, 1732, 2638, 2658,
..... 3026, 3586, 3604, 4089, 4648, 4741, 5521,
..... 5531, 5565, 5613, 5652, 6508, 6514, 6520, 6877
\l_@@_first_i_tl ..... 6803,
..... 6809, 6810, 6840, 6871, 6880, 6882, 6937,
..... 6944, 6946, 7028, 7039, 7043, 7139, 7160, 7188
\l_@@_first_j_tl .....
..... 6804, 6812, 6813, 6844, 6846, 6907, 6920,
..... 6927, 6929, 7029, 7040, 7044, 7142, 7144, 7154
\l_@@_first_row_int .. 330, 331, 577, 855,
..... 1244, 1331, 1404, 1617, 2420, 2499, 2527,
..... 2538, 3023, 3456, 3478, 5514, 5528, 5555,
..... 5612, 5650, 6147, 6165, 6506, 6666, 6841, 7454
\c_@@_footnote_bool .....
..... 1491, 1686, 7251, 7273, 7296, 7299, 7309, 7315
\c_@@_footnotehyper_bool .. 7250, 7274, 7306
\c_@@_forbidden_letters_str .... 5218, 5246
\@@_full_name_env: .....
..... 261, 7351, 7358, 7366, 7374, 7378, 7468,
..... 7486, 7512, 7525, 7530, 7535, 7537, 7567,
..... 7586, 7591, 7596, 7602, 7611, 7650, 7659, 7875
\@@_hdottedline: ..... 1224, 5449
\@@_hdottedline:n ..... 5457, 5459
\@@_hdottedline_i: ..... 5452, 5454
\@@_hline:n 4966, 5158, 5178, 5271, 5460, 6379
\@@_hline_i: ..... 4972, 4975
\@@_hline_ii: ..... 5000, 5008, 5036
\@@_hline_iii: ..... 5044, 5048
\@@_hline_iv: ..... 5041, 5094
\@@_hline_v: ..... 5045, 5126
\@@_hlines_block:nnn ..... 6045, 6365
\l_@@_hlines_block_bool 320, 5962, 6041, 6052
\l_@@_hlines_clist .. 325, 588, 602, 607,
..... 638, 1120, 1122, 1126, 1166, 3103, 5156, 5157
\l_@@_hpos_block_str .... 313, 314, 5696,
..... 5698, 5700, 5702, 5704, 5706, 5743, 5744,
..... 5746, 5798, 5810, 5823, 5834, 5885, 5909,
..... 5913, 5926, 5931, 5966, 5968, 5970, 5972,
..... 5975, 5978, 6217, 6229, 6240, 6244, 6254, 6266
\l_@@_hpos_cell_str ..... 234, 235,
..... 1825, 1921, 1923, 2027, 2302, 2345, 5742, 5744
\l_@@_hpos_col_str .....
..... 1877, 1879, 1881, 1883, 1908, 1920,
..... 1924, 1926, 1934, 1935, 1938, 1943, 2010, 2161
\l_@@_hpos_of_block_cap_bool .....
..... 315, 5973, 5976, 5979, 6144, 6214, 6251
\g_@@_ht_last_row_dim .....
..... 916, 1204, 1205, 1279, 1280, 1625
\g_@@_ht_row_one_dim .. 942, 943, 1201, 1202
\g_@@_ht_row_zero_dim .....
..... 936, 937, 1199, 1200, 1620, 2501, 2540
\@@_i: ..... 5514, 5516,
..... 5517, 5518, 5519, 5528, 5534, 5536, 5537,
..... 5538, 5539, 5544, 5545, 5546, 5547, 5555,
..... 5558, 5560, 5561, 5562, 5614, 5616, 5619,
..... 5620, 5624, 5625, 5650, 5655, 5657, 5659,
..... 5663, 5664, 5675, 5682, 5684, 5686, 5690, 5691
\g_@@_iddots_int ..... 3082, 3769, 3770
\l_@@_in_env_bool .... 241, 402, 1505, 1506
\c_@@_in_preamble_bool .. 21, 22, 24, 698, 714
\l_@@_initial_i_int ..... 3088,
..... 3247, 3322, 3325, 3350, 3358, 3362, 3371,
..... 3379, 3389, 3436, 3521, 3567, 3569, 3583,
..... 3589, 3592, 4128, 4129, 4139, 4207, 4217, 4219
\l_@@_initial_j_int .....
..... 3089, 3248, 3323, 3330,
..... 3335, 3341, 3351, 3359, 3363, 3372, 3380,
..... 3390, 3392, 3437, 3459, 3462, 3470, 3657,
..... 3659, 3664, 4132, 4142, 4144, 4203, 4204, 4215
\l_@@_initial_open_bool .....
..... 3092, 3324, 3328, 3331,
..... 3338, 3344, 3348, 3364, 3518, 3557, 3564,
..... 3574, 3640, 3647, 3653, 3707, 3755, 3895,
..... 3942, 4126, 4133, 4145, 4201, 4208, 4220, 4278
\@@_insert_tabularnotes: ..... 2451, 2454
\@@_instruction_of_type:nnn .....
..... 1063, 3985, 4001, 4017, 4040, 4064
\c_@@_integers_alist_tl ..... 7013, 7024
\l_@@_inter_dots_dim .....
..... 486, 488, 3097, 3900, 3907,
..... 3918, 3926, 3933, 3938, 3950, 3958, 5110, 5120

```

```

\g_@@_internal_code_after_tl .....
..... 269, 1861, 2089, 2110,
2118, 2131, 2137, 2143, 2204, 3122, 3123,
5176, 5270, 5290, 5456, 5774, 6106, 6561, 6784
\@@_intersect_our_row:nnnnn ..... 4610
\@@_intersect_our_row_p:nnnnn ..... 4560
\@@_j: ..... 5521, 5523,
5524, 5525, 5526, 5531, 5534, 5536, 5539,
5541, 5542, 5544, 5547, 5549, 5550, 5565,
5568, 5570, 5571, 5572, 5627, 5629, 5632,
5634, 5638, 5639, 5652, 5655, 5656, 5658,
5663, 5664, 5676, 5682, 5683, 5685, 5690, 5691
\l_@@_key_nb_rows_int .....
..... 233, 4329, 4349, 4357, 4364
\l_@@_l_dim .....
.... 3873, 3874, 3887, 3888, 3900, 3906,
3917, 3925, 3933, 3938, 3950, 3951, 3958, 3959
\l_@@_large_nodes_bool 514, 615, 5506, 5510
\g_@@_last_col_found_bool ..... 340,
1249, 1594, 1662, 2727, 2803, 2878, 3013, 6241
\l_@@_last_col_int .....
.. 338, 339, 787, 823, 825, 838, 854, 875,
1270, 1273, 1597, 1744, 2950, 2952, 3014,
3016, 3627, 3662, 3982, 3998, 4036, 4060,
6000, 6005, 6006, 6007, 6010, 6038, 6048,
6064, 6076, 6088, 6100, 6112, 6127, 6168,
6173, 6181, 6197, 6510, 6516, 6522, 7329, 7352
\l_@@_last_col_without_value_bool ...
..... 337, 822, 3015, 7332
\l_@@_last_empty_column_int .....
..... 5385, 5386, 5399, 5405, 5418
\l_@@_last_empty_row_int .....
..... 5367, 5368, 5381, 5402
\l_@@_last_i_tl .....
.... 6805, 6816, 6817, 6827, 6840, 6874,
6887, 6889, 6937, 6944, 7124, 7139, 7160, 7220
\l_@@_last_j_tl .....
..... 6806, 6819, 6820, 6828, 6851,
6853, 6910, 6920, 6927, 7125, 7163, 7165, 7175
\l_@@_last_row_int .....
334, 335, 578, 904, 950, 1132, 1264, 1268,
1275, 1582, 1586, 1589, 1601, 1623, 2610,
2611, 2842, 2843, 2887, 2888, 3018, 3507,
3546, 4014, 4030, 4054, 4747, 4755, 5999,
6002, 6003, 6022, 6038, 6048, 6064, 6076,
6088, 6099, 6111, 6125, 6196, 6207, 6518, 7649
\l_@@_last_row_without_value_bool ...
..... 336, 1266, 1584, 3017
\l_@@_left_delim_dim .....
..... 1294, 1298, 1303, 2576, 2865
\g_@@_left_delim_tl .....
1302, 1493, 1634, 1658, 1728, 2074, 2076, 5109
\l_@@_left_margin_dim .....
..... 516, 618, 1308, 2866, 5106, 5643
\l_@@_letter_for_dotted_lines_str ...
..... 803, 814, 815, 1794, 7605
\l_@@_letter_str ..... 5190,
5209, 5213, 5215, 5218, 5223, 5249, 7496, 7507
\l_@@_letter_vlism_tl ..... 255, 595, 1795
\l_@@_light_syntax_bool 498, 572, 1311, 1544
\@@_light_syntax_i ..... 2601, 2604
\@@_line ..... 3121, 4262
\@@_line_i:nn ..... 4269, 4276

\l_@@_line_width_dim .....
312, 5964, 6289, 6327, 6338, 6344, 6367,
6390, 6410, 6425, 6427, 6437, 6438, 6440, 6451
\@@_line_with_light_syntax:n ... 2615, 2619
\@@_line_with_light_syntax:i:n .....
..... 2614, 2620, 2621
\l_@@_local_end_int .....
..... 4797, 4818, 4826, 4876, 4925,
4940, 4978, 4999, 5007, 5057, 5112, 5114, 5135
\l_@@_local_start_int .....
..... 4796, 4812, 4813, 4816,
4820, 4824, 4872, 4923, 4936, 4977, 4993,
4994, 4997, 5001, 5005, 5053, 5102, 5104, 5131
\@@_math_toggle_token: ..... 183, 984,
2836, 2853, 2882, 2898, 6606, 6617, 7206, 7242
\g_@@_max_cell_width_dim .....
..... 993, 994, 1516, 2702, 5469, 5495
\c_@@_max_l_dim ..... 3887, 3892
\l_@@_medium_nodes_bool 513, 614, 5504, 6188
\@@_message_hdotsfor: 7338, 7351, 7358, 7366
\@@_msg_new:nn ..... 17,
7278, 7287, 7343, 7348, 7355, 7363, 7371,
7376, 7382, 7387, 7392, 7397, 7402, 7407,
7412, 7417, 7423, 7429, 7434, 7440, 7446,
7451, 7458, 7465, 7471, 7480, 7485, 7487,
7494, 7499, 7504, 7510, 7516, 7533, 7540,
7545, 7552, 7559, 7565, 7572, 7579, 7584,
7594, 7600, 7608, 7615, 7622, 7628, 7634,
7641, 7647, 7655, 7657, 7663, 7668, 7685, 7990
\@@_msg_new:nnn .....
..... 18, 7252, 7522, 7674, 7691, 7703,
7718, 7739, 7757, 7774, 7819, 7872, 7924, 7976
\@@_msg_redirect_name:nn .....
..... 19, 798, 1674, 6013, 6016
\@@_multicolumn:nnn ..... 1239, 2234
\g_@@_multicolumn_cells_seq .....
..... 298, 1242, 1289, 2249,
3039, 3043, 3044, 5539, 5547, 5669, 6152, 6170
\g_@@_multicolumn_sizes_seq .....
..... 299, 1243, 1290, 2251, 3045, 3046, 5670
\l_@@_multiplicity_int .....
... 4769, 4880, 4892, 4902, 5061, 5071, 5081
\g_@@_name_env_str ..... 259, 264,
265, 1500, 1501, 2631, 2922, 2923, 2931,
2932, 2961, 2974, 2993, 3003, 3141, 6531, 7327
\l_@@_name_str .....
... 512, 668, 922, 925, 1025, 1028, 1103,
1106, 2648, 2649, 2673, 2674, 2691, 2692,
2722, 2723, 2748, 2751, 2796, 2799, 2817,
2820, 3063, 3068, 3074, 3221, 3222, 3233,
3236, 3239, 5660, 5663, 5687, 5690, 6137, 6140
\g_@@_names_seq ..... 240, 665, 667, 7988
\l_@@_nb_cols_int .....
..... 6484, 6495, 6503, 6509, 6515, 6521
\l_@@_nb_rows_int ..... 6483, 6494, 6512
\@@_newcolumnntype 1139, 1693, 1694, 1695, 2980
\@@_node_for_cell: .....
..... 1002, 1009, 1377, 2863, 2913
\@@_node_for_multicolumn:nn .... 5671, 5678
\@@_node_left:nn ..... 6986, 6987, 7047
\@@_node_position: .....
1333, 1335, 1339, 1341, 1406, 1408, 1416, 1422
\@@_node_position_i: ..... 1419, 1423

```


\@@_node_right:nnnn	6996, 6998, 7071	\@@_patch_preamble_ii:nn	1769, 1770, 1771, 1832
\g_@@_not_empty_cell_bool	278, 1000, 1007, 2969	\@@_patch_preamble_iii:n	1772, 1837, 1845
\@@_not_in_exterior:nnnnn	4602	\@@_patch_preamble_iii_i:n	1840, 1842
\@@_not_in_exterior_p:nnnnn	4538	\@@_patch_preamble_iv:n	1773, 1774, 1775, 1893
\l_@@_notes_above_space_dim	505, 507	\@@_patch_preamble_iv_i:n	1896, 1898
\l_@@_notes_bottomrule_bool	686, 841, 869, 2477	\@@_patch_preamble_iv_ii:w	1901, 1902, 1904
\l_@@_notes_code_after_tl	684, 2486	\@@_patch_preamble_iv_iii:nn	1905, 1906
\l_@@_notes_code_before_tl	682, 2458	\@@_patch_preamble_iv_iv:nn	1910, 1912, 2013, 2016, 2176
\@@_notes_label_in_list:n	371, 390, 398, 694	\@@_patch_preamble_iv_v:nnnnnnnn	1916, 1949
\@@_notes_label_in_tabular:n	370, 411, 691	\@@_patch_preamble_ix:nn	1784, 1785, 1786, 2098
\l_@@_notes_para_bool	680, 839, 867, 2462	\@@_patch_preamble_ix_i:nnn	2102, 2124
\@@_notes_style:n	369, 372, 390, 398, 414, 419, 688	\@@_patch_preamble_v:n	1776, 1777, 1999
\l_@@_nullify_dots_bool	508, 613, 3989, 4005, 4021, 4045, 4068	\@@_patch_preamble_v_i:w	2002, 2003, 2005
\l_@@_number_of_notes_int	368, 405, 415, 425	\@@_patch_preamble_v_ii:nn	2006, 2007
\@@_old_CT@arc@	1507, 3143	\@@_patch_preamble_vi:nnnn	1778, 1779, 2019
\@@_old_cdots	1212, 4006	\@@_patch_preamble_vii:n	1780, 2042
\@@_old_ddots	1214, 4046	\@@_patch_preamble_vii_i:w	2045, 2046, 2048
\@@_old_dotfill	6547, 6550, 6558	\@@_patch_preamble_vii_ii:n	2049, 2050
\@@_old_dotfill:	1232	\@@_patch_preamble_viii:nn	1781, 1782, 1783, 2069
\l_@@_old_iRow_int	270, 1260, 3190	\@@_patch_preamble_viii_i:nn	2082, 2085, 2087
\@@_old_ialign:	1085, 1208, 3117, 5994	\@@_patch_preamble_x:n	1787, 1788, 2149
\@@_old_iddots	1215, 4069	\@@_patch_preamble_x_i:w	2152, 2153, 2155
\l_@@_old_jCol_int	271, 1262, 3191	\@@_patch_preamble_x_ii:n	2156, 2159
\@@_old_ldots	1211, 3990	\@@_patch_preamble_xi:n	1830, 1947, 2040, 2065, 2197, 2208, 2232
\@@_old_multicolumn	1241, 4084	\@@_patch_preamble_xii:n	1794, 2200
\@@_old_pgfpntanchor	197, 7005, 7009	\@@_patch_preamble_xiii:n	2211, 2229
\@@_old_pgful@check@rerun	102, 106	\@@_pgf_rect_node:nnn	458, 1420, 6190
\@@_old_vdots	1213, 4022	\@@_pgf_rect_node:nnnnn	433, 1462, 5654, 5681, 6129, 6184, 6973
\@@_open_x_final_dim:	3475, 3527, 3561, 3715, 3764	\c_@@_pgfortikzpicture_tl	51, 55, 3172, 4295
\@@_open_x_initial_dim:	3453, 3520, 3558, 3710, 3758	\@@_pgfpntanchor:n	7001, 7006
\@@_open_y_final_dim:	3599, 3651, 3763	\@@_pgfpntanchor_i:nn	7009, 7011
\@@_open_y_initial_dim:	3581, 3648, 3709, 3757	\@@_pgfpntanchor_ii:w	7012, 7020
\l_@@_other_keys_tl	4789, 4858, 4971, 5039	\@@_pgfpntanchor_iii:w	7033, 7035
\@@_overbrace_i:n	7181, 7186	\@@_picture_position:	1328, 1335, 1341, 1408, 1422, 1423
\l_@@_parallelize_diags_bool	502, 503, 610, 3079, 3719, 3767	\g_@@_pos_of_blocks_seq	292, 1288, 1446, 1518, 2252, 3031, 3035, 3036, 3149, 4536, 4803, 4984, 5432, 6054, 6571
\@@_patch_for_revtext:	1470, 1489	\g_@@_pos_of_stroken_blocks_seq	294, 1519, 4807, 4988, 6066
\@@_patch_m_preamble:n	2243, 2278, 2311, 2316, 2377	\g_@@_pos_of_xdots_seq	293, 1520, 3387, 4805, 4986
\@@_patch_m_preamble_i:n	2282, 2283, 2284, 2298	\l_@@_position_int	4759, 4790, 4798, 4874, 4920, 4938, 4979, 5055, 5099, 5133
\@@_patch_m_preamble_ii:nn	2285, 2286, 2287, 2308	\g_@@_post_action_cell_tl	882, 986, 1980, 2186, 4370, 4381
\@@_patch_m_preamble_iii:n	2288, 2313	\@@_pre_array:	1258, 1319, 1541
\@@_patch_m_preamble_iv:nnn	2289, 2290, 2291, 2318	\@@_pre_array_i:w	1315, 1541
\@@_patch_m_preamble_ix:n	2362, 2380	\@@_pre_array_ii:	1162, 1291
\@@_patch_m_preamble_v:nnnn	2292, 2293, 2338	\@@_pre_code_before:	1321, 1398
\@@_patch_m_preamble_x:n	2306, 2336, 2357, 2359, 2383	\c_@@_preamble_first_col_tl	1733, 2828
\@@_patch_node_for_cell:	1035, 1377	\c_@@_preamble_last_col_tl	1745, 2873
\@@_patch_node_for_cell:n	1033, 1059, 1062	\g_@@_preamble_tl	1495, 1696, 1700, 1704, 1710, 1724, 1733, 1742, 1745, 1754, 1758, 1799, 1807,
\@@_patch_preamble:n	1716, 1762, 1801, 1808, 1816, 1835, 1871, 2078, 2094, 2096, 2112, 2120, 2146, 2206, 2226		
\@@_patch_preamble_i:n	1766, 1767, 1768, 1821		

1823, 1834, 1847, 1951, 2021, 2054, 2081,
2109, 2117, 2130, 2136, 2181, 2202, 2215,
2222, 2231, 2242, 2244, 2300, 2310, 2315,
2320, 2340, 2366, 2373, 2382, 2585, 2612, 5285
\@@_pred:n
144, 182, 2952, 4836, 4849, 5017, 5030, 6360
\@@_provide_pgfsyspdfmark: ... 69, 78, 1490
\@@_put_box_in_flow: 1660, 2385, 2578
\@@_put_box_in_flow_bis:nn 1657, 2545
\@@_put_box_in_flow_i: 2391, 2393
\@@_qpoint:n 225, 1454, 1456,
1458, 1460, 2396, 2398, 2410, 2426, 2493,
2495, 2511, 2522, 2533, 3209, 3211, 3213,
3215, 3225, 3227, 3470, 3492, 3521, 3528,
3567, 3569, 3583, 3601, 3657, 3659, 4303,
4306, 4647, 4651, 4667, 4669, 4872, 4874,
4876, 4920, 4923, 4925, 4936, 4938, 4940,
5053, 5055, 5057, 5099, 5102, 5112, 5131,
5133, 5135, 5560, 5570, 6121, 6123, 6125,
6127, 6161, 6181, 6210, 6311, 6313, 6320,
6322, 6424, 6426, 6428, 6435, 6439, 6441,
6585, 6587, 6590, 6592, 6659, 6661, 6871,
6874, 6912, 6929, 6946, 7154, 7175, 7188, 7220
\l_@@_radius_dim
492, 494, 2203, 3096, 3532, 3533, 3967, 5451
\l_@@_real_left_delim_dim 2547, 2562, 2577
\l_@@_real_right_delim_dim 2548, 2574, 2580
\@@_recreate_cell_nodes: 1344, 1402
\g_@@_recreate_cell_nodes_bool
..... 511, 1177, 1344, 1369, 1376, 1381
\@@_rectanglecolor
..... 1356, 4344, 4461, 4494, 4511, 4721
\@@_rectanglecolor:nnn ... 4467, 4476, 4479
\@@_renew_NC@rewrite@S: ... 208, 210, 1248
\@@_renew_dots: 1146, 1238
\l_@@_renew_dots_bool 611, 1238, 7267
\@@_renew_matrix: 790, 6463, 7269
\l_@@_respect_arraystretch_bool
509, 629, 5714, 5794, 5805, 5904, 5921, 5988
\l_@@_respect_blocks_bool 4517, 4534, 4557
\@@_restore_iRow_jCol: 3142, 3188
\c_@@_revtex_bool 59, 61, 64, 66, 1489
\l_@@_right_delim_dim
..... 1295, 1299, 1305, 2579, 2910
\g_@@_right_delim_tl .. 1304, 1494, 1652,
1658, 1729, 2077, 2106, 2107, 2128, 2133, 5119
\l_@@_right_margin_dim
..... 517, 620, 1548, 2911, 3666, 5116, 5646
\@@_rotate: 1230, 4255
\g_@@_rotate_bool
.. 246, 959, 987, 1971, 2034, 2352, 2855,
2900, 4255, 5798, 5842, 5847, 5908, 5925, 6117
\@@_rotate_cell_box:
..... 947, 987, 2034, 2352, 2855, 2900, 6117
\l_@@_rounded_corners_dim
310, 5953, 6089, 6303, 6304, 6339, 6392, 6449
\@@_roundedrectanglecolor 1357, 4470
\l_@@_row_max_int 301, 3257, 3400, 3417
\l_@@_row_min_int 300, 3325, 3398, 3415
\g_@@_row_of_col_done_bool
..... 277, 1117, 1499, 2657
\g_@@_row_style_tl
..... 281, 894, 1522, 1963, 4362, 4363,
4365, 4368, 4379, 4390, 4395, 4406, 4407, 5792
\g_@@_row_total_int 231, 1245, 1325,
1331, 1404, 1600, 2421, 2528, 3018, 3025,
3456, 3478, 4170, 5514, 5528, 5555, 5650,
6022, 6147, 6165, 6666, 6827, 6841, 7124, 7455
\@@_rowcolor 1358, 4353, 4443
\@@_rowcolor_tabular 1159, 4715
\@@_rowcolors 1359, 4595
\@@_rowcolors_i:nnnnn 4561, 4597
\l_@@_rowcolors_restart_bool ... 4520, 4549
\@@_rowlistcolors 1360, 4524, 4596
\g_@@_rows_seq . 2607, 2609, 2611, 2613, 2615
\l_@@_rows_tl
..... 4439, 4487, 4563, 4580, 4628, 4653
\l_@@_rule_width_dim 4781,
4939, 5134, 5191, 5234, 5242, 5259, 5279, 5288
\l_@@_rules_color_tl
..... 272, 547, 1539, 1540, 6902, 6903
\@@_set_CT@arc@: 185, 1540, 4774, 6903
\@@_set_CT@arc@_i: 186, 187
\@@_set_CT@arc@_ii: 186, 189
\@@_set_CT@drsc@: 191, 4776
\@@_set_CT@drsc@_i: 192, 193
\@@_set_CT@drsc@_ii: 192, 195
\@@_set_final_coords: 3426, 3451
\@@_set_final_coords_from_anchor:n ..
... 3442, 3531, 3562, 3643, 3652, 3718, 3766
\@@_set_initial_coords: 3421, 3440
\@@_set_initial_coords_from_anchor:n ..
... 3431, 3524, 3559, 3642, 3649, 3712, 3760
\@@_set_size:n 6481, 6496
\c_@@_siunitx_loaded_bool ... 198, 202, 207
\c_@@_size_seq
... 1268, 1273, 1323, 1324, 1325, 1326, 3021
\l_@@_small_bool 788, 828, 834,
856, 891, 1172, 2071, 2100, 2837, 2883, 3094
\@@_standard_cline 140, 1217
\@@_standard_cline:w 140, 141
\l_@@_standard_cline_bool .. 483, 559, 1216
\c_@@_standard_tl 496, 497, 3793
\l_@@_start_int 4761, 4799, 4980
\g_@@_static_num_of_col_int
..... 305, 1669, 1717, 6010, 7367, 7379, 7587
\l_@@_stop_loop_bool 3251, 3252,
3284, 3297, 3306, 3319, 3320, 3352, 3365, 3374
\@@_store_in_tmpb_tl 6281, 6283
\@@_stroke_block:nnn 6061, 6285
\@@_stroke_borders_block:nnn ... 6073, 6388
\@@_stroke_horizontal:n .. 6416, 6418, 6433
\@@_stroke_vertical:n 6412, 6414, 6422
\@@_sub_matrix:nnnnnnn 6796, 6822
\@@_sub_matrix_i:nnnn 6863, 6869
\l_@@_submatrix_extra_height_dim
..... 322, 6712, 6897
\l_@@_submatrix_hlines_clist
..... 327, 6724, 6744, 6936, 6938
\l_@@_submatrix_left_xshift_dim
..... 323, 6714, 6949, 6982
\l_@@_submatrix_name_str
6759, 6831, 6971, 6973, 6985, 6987, 6995, 6999

<code>\g_@@_submatrix_names_seq</code>	<code>\g_@@_total_X_weight_int</code>
..... 296, 3125, 6756, 6760, 7531 273, 1165, 1553, 1556, 1575, 2173
<code>\l_@@_submatrix_right_xshift_dim</code>	<code>\l_@@_type_of_col_tl</code>
..... 324, 6716, 6958, 6992	826, 827, 2962, 2964, 6488, 6489, 6490, 6498, 6503
<code>\g_@@_submatrix_seq</code> ... 304, 1286, 3402, 6782	<code>\c_@@_types_of_matrix_seq</code>
<code>\l_@@_submatrix_slim_bool</code> 6722, 6839, 7476 7317, 7318, 7323, 7327
<code>\l_@@_submatrix_vlines_clist</code>	<code>\@@_underbrace_i:n</code>
..... 328, 6726, 6746, 6919, 6921	7180, 7218
<code>\@@_succ:n</code> ... 177, 181, 1095, 1101, 1106,	<code>\@@_update_for_first_and_last_row:</code> ..
1107, 1127, 1458, 1460, 1865, 2090, 2220, 930, 995, 1277, 2857, 2902
2250, 2371, 2398, 2740, 2746, 2751, 2752,	<code>\@@_use_arraybox_with_notes:</code> ... 1614, 2506
2778, 2786, 2799, 2800, 2811, 2815, 2820,	<code>\@@_use_arraybox_with_notes_b:</code> . 1611, 2490
2821, 3492, 3569, 3659, 4606, 4651, 4667,	<code>\@@_use_arraybox_with_notes_c:</code>
4832, 4957, 5013, 5152, 5291, 5437, 5439, 1612, 1643, 2434, 2504, 2543
5441, 5443, 5620, 5624, 5634, 5638, 6125,	<code>\c_@@_varwidth_loaded_bool</code> ... 30, 34, 2012
6127, 6181, 6320, 6322, 6459, 6590, 6592, 6661	<code>\@@_vdottedline:n</code>
<code>\l_@@_suffix_tl</code>	2205, 5461
5582, 5593, 5603, 5606, 5655, 5663, 5664, 5682, 5690, 5691	<code>\@@_vline:n</code> 1863, 4784, 4963, 5291, 5462, 6356
<code>\l_@@_tabular_width_dim</code>	<code>\@@_vline_i:</code>
..... 244, 1079, 1081, 1756, 3004	4791, 4794
<code>\l_@@_tabularnote_tl</code> 367, 843, 871, 2450, 2459	<code>\@@_vline_ii:</code>
<code>\g_@@_tabularnotes_seq</code>	4819, 4827, 4855
..... 366, 406, 2465, 2471, 2487	<code>\@@_vline_iii:</code>
<code>\c_@@_tabularx_loaded_bool</code> ... 28, 46, 2991	4863, 4867
<code>\@@_test_hline_in_block:nnnnn</code>	<code>\@@_vline_iv:</code>
..... 4985, 4987, 5293	4860, 4915
<code>\@@_test_hline_in_stroken_block:nnnn</code> .	<code>\@@_vline_v:</code>
..... 4989, 5315	4864, 4931
<code>\@@_test_if_cell_in_a_block:nn</code>	<code>\@@_vlines_block:nnn</code>
..... 5371, 5389, 5407, 5427	6035, 6342
<code>\@@_test_if_cell_in_block:nnnnnnn</code> ...	<code>\l_@@_vlines_block_bool</code> 319, 5960, 6031, 6052
..... 5433, 5435	<code>\l_@@_vlines_clist</code>
<code>\@@_test_if_math_mode:</code> 249, 1504, 2933	326, 589, 601,
<code>\@@_test_in_corner_h:</code>	606, 637, 1167, 1702, 1708, 1739, 1751,
4990, 5011	2213, 2220, 2364, 2371, 2766, 3104, 4961, 4962
<code>\@@_test_in_corner_v:</code>	<code>\l_@@_vpos_col_str</code>
4809, 4830	1885, 1888, 1890, 1895, 1917, 1933, 2009, 2162
<code>\@@_test_vline_in_block:nnnnn</code>	<code>\l_@@_vpos_of_block_tl</code> .. 316, 317, 5708,
..... 4804, 4806, 5304	5710, 5808, 5822, 5833, 5912, 5930, 5981, 5983
<code>\@@_test_vline_in_stroken_block:nnnn</code> .	<code>\@@_w:</code>
..... 4808, 5326	1693, 1778, 2292
<code>\l_@@_the_array_box</code>	<code>\l_@@_weight_int</code>
1292, 1307, 1566, 1574, 2438, 2439, 2441, 2444	2158, 2163, 2164, 2167, 2170, 2171, 2173, 2177
<code>\c_@@_tikz_loaded_bool</code>	<code>\l_@@_width_dim</code>
..... 29, 50, 1347, 3105, 5945	239,
<code>\l_@@_tikz_rule_tl</code>	783, 864, 1566, 1574, 2972, 2973, 2994, 2995
... 4778, 4862, 4942, 4943, 5043, 5137, 5138	<code>\g_@@_width_first_col_dim</code>
<code>\l_@@_tikz_seq</code> 290, 1498, 1605, 2652, 2858, 2859
308, 5946, 6092, 6101	<code>\g_@@_width_last_col_dim</code>
<code>\l_@@_tmpc_dim</code> 289, 1497, 1664, 2807, 2903, 2904
286, 1459,	<code>\l_@@_width_used_bool</code>
1466, 3214, 3218, 4649, 4650, 4673, 4877,	297, 865, 1551
4888, 4896, 4901, 4907, 4941, 4945, 5058,	<code>\l_@@_x_final_dim</code>
5075, 5080, 5086, 5136, 5140, 6126, 6131, 284, 3428, 3477, 3486, 3487, 3490,
6186, 6314, 6325, 6427, 6430, 6591, 6594, 6602	3493, 3494, 3645, 3661, 3669, 3673, 3677,
<code>\l_@@_tmpc_int</code>	3679, 3684, 3686, 3716, 3725, 3733, 3773,
4552, 4553, 4554, 5236, 5237, 5253, 5281, 5282	3781, 3822, 3837, 3846, 3880, 3932, 3948,
<code>\l_@@_tmpc_tl</code> ... 4332, 4340, 4345, 4354,	4307, 4922, 5113, 5116, 5118, 5120, 6838,
4482, 4484, 4487, 4646, 4665, 6347, 6359,	6854, 6855, 6861, 6958, 6975, 6992, 7159,
6370, 6375, 6401, 6418, 6424, 6777, 6779, 6783	7166, 7167, 7173, 7176, 7192, 7209, 7224, 7239
<code>\l_@@_tmpd_dim</code>	<code>\l_@@_x_initial_dim</code> 282, 3423, 3455,
287,	3464, 3465, 3468, 3471, 3472, 3645, 3660,
1461, 1467, 3216, 3219, 4670, 4673, 4889,	3661, 3668, 3673, 3677, 3679, 3681, 3684,
4896, 5068, 5075, 6128, 6131, 6164, 6175,	3686, 3725, 3733, 3773, 3781, 3819, 3836,
6179, 6182, 6186, 6323, 6326, 6593, 6594, 6612	3846, 3880, 3932, 3946, 3948, 3966, 3968,
<code>\l_@@_tmpd_tl</code> 4483, 4485, 4488, 6348, 6352,	4304, 4921, 5103, 5106, 5108, 5110, 6837,
6371, 6382, 6402, 6414, 6435, 6778, 6780, 6783	6847, 6848, 6858, 6949, 6974, 6982, 7138,
	7145, 7146, 7152, 7155, 7192, 7209, 7224, 7239
	<code>\l_@@_xdots_color_tl</code> 522, 536, 3511, 3550,
	3631, 3632, 3700, 3748, 3802, 4174, 4249, 4266
	<code>\l_@@_xdots_down_tl</code> ... 540, 3809, 3830, 3865
	<code>\l_@@_xdots_line_style_tl</code>
 495, 497, 532, 3793, 3802
	<code>\l_@@_xdots_shorten_dim</code>
	489,
	491, 538, 3098, 3816, 3817, 3906, 3917, 3925

$\backslash l_@_x_dots_up_tl$	541, 3808, 3829, 3855	$\backslash begingroup$	2237
$\backslash l_@_y_final_dim$	285, 3429, 3529, 3533, 3571, 3575, 3577, 3602, 3612, 3613, 3727, 3730, 3775, 3778, 3822, 3837, 3845, 3882, 3937, 3956, 4308, 4926, 5101, 6662, 6684, 6699, 6875, 6890, 6891, 6896, 6914, 6931, 6975, 6983, 6993	$\backslash bfseries$	4399, 4402
$\backslash l_@_y_initial_dim$	283, 3424, 3522, 3532, 3570, 3571, 3575, 3577, 3584, 3594, 3595, 3727, 3732, 3775, 3780, 3819, 3836, 3845, 3882, 3937, 3954, 3956, 3966, 3969, 4305, 4924, 5100, 6660, 6684, 6699, 6872, 6883, 6884, 6896, 6913, 6930, 6974, 6983, 6993	$\backslash bgroup$	1492
$\backslash \backslash$	2598, 2620, 6510, 6516, 6522, 6640, 7254, 7255, 7284, 7293, 7379, 7384, 7389, 7394, 7399, 7437, 7442, 7448, 7455, 7461, 7462, 7477, 7482, 7491, 7496, 7507, 7513, 7519, 7525, 7526, 7537, 7549, 7556, 7568, 7575, 7582, 7591, 7597, 7605, 7612, 7619, 7625, 7631, 7656, 7660, 7665, 7671, 7677, 7688, 7694, 7695, 7705, 7706, 7720, 7721, 7737, 7741, 7742, 7760, 7761, 7777, 7778, 7822, 7823, 7875, 7876, 7927, 7928, 7983, 7984	$\backslash BNiceMatrix$	6478
$\backslash \{$	265, 1783, 2091, 2116, 2940, 6539, 6954, 7518, 7612, 7822, 7927	$\backslash bNiceMatrix$	6475
$\backslash \}$	265, 1786, 2091, 2101, 2940, 6539, 6963, 7518, 7612, 7822, 7927	$\backslash Body$	1315
$\backslash $	2942, 6538	$\backslash boldmath$	4399, 4402
$\backslash \sqcup$	7341, 7351, 7358, 7366, 7367, 7378, 7379, 7454, 7455, 7468, 7474, 7478, 7486, 7512, 7524, 7530, 7542, 7586, 7587, 7588, 7596, 7602, 7610, 7617, 7630, 7650, 7651, 7659, 7694	bool commands:	
A		$\backslash bool_do_until:Nn$	3252, 3320
$\backslash A$	6754	$\backslash bool_gset_false:N$	959, 1006, 1007, 1116, 1249, 1369, 1499, 1523, 1701, 2869, 2916, 5302, 5313, 5324, 5335, 5847
$\backslash aboverulesep$	2481	$\backslash bool_gset_true:N$	1526, 1870, 2657, 2833, 2878, 2969, 3991, 4007, 4023, 4047, 4070, 4081, 4255, 4802, 4983
$\backslash addtocounter$	423	$\backslash bool_if:N\TF$	184, 702, 707, 718, 723, 888, 891, 987, 1090, 1117, 1168, 1172, 1177, 1237, 1238, 1267, 1272, 1287, 1344, 1347, 1371, 1374, 1376, 1396, 1489, 1491, 1505, 1515, 1551, 1584, 1662, 1667, 1686, 1691, 1719, 1731, 1971, 2034, 2071, 2100, 2352, 2477, 2643, 2660, 2678, 2696, 2709, 2735, 2771, 2803, 2808, 2837, 2855, 2883, 2900, 2991, 3013, 3015, 3017, 3079, 3094, 3105, 3574, 3576, 3719, 3767, 3989, 4005, 4021, 4045, 4068, 4393, 4534, 4557, 5107, 5117, 5196, 5203, 5205, 5232, 5277, 5380, 5398, 5416, 5478, 5488, 5510, 5794, 5798, 5842, 5904, 5908, 5921, 5925, 6031, 6041, 6117, 6144, 6188, 6214, 6251, 6551, 7136, 7157, 7296, 7306, 7332, 7476
$\backslash alph$	369	$\backslash bool_if:n\TF$	207, 375, 402, 1065, 1594, 3407, 4280, 4612, 4952, 4956, 5147, 5151, 5504, 5755, 6051, 6241, 6663, 6673, 6675, 6688, 6694, 6703
$\backslash anchor$	3200, 3201	$\backslash bool_lazy_all:n\TF$	1735, 1747, 2762, 4878, 5059, 5295, 5306, 5317, 5328, 5801
$\backslash array$	1478	$\backslash bool_lazy_and:nn\TF$	2436, 2699, 2776, 2780, 2788, 2838, 3563, 3828, 4087, 4623, 5208, 6307, 6923, 6940
$\backslash arraybackslash$	1964, 2188, 2326	$\backslash bool_lazy_or:nn\TF$	529, 999, 1057, 1727, 2419, 2448, 2526, 2886, 3640, 3792, 3886, 4604, 4636, 4640, 4690, 4694, 5372, 5390, 5408, 5730, 5735, 6826, 7123, 7150, 7171
$\backslash arraycolor$	1361	$\backslash bool_lazy_or:p:nn$	2841
$\backslash arraycolsep$	619, 621, 623, 1078, 1175, 1298, 1299, 1642, 1646, 2439, 2777, 2781, 2791, 5108, 5118	$\backslash bool_not_p:n$	1738, 1740, 1750, 1752, 2701, 2765, 2767, 7151, 7172
$\backslash arrayrulecolor$	116	$\backslash bool_set:Nn$	3644
$\backslash arrayrulewidth$	148, 153, 173, 549, 921, 1094, 1096, 1102, 1133, 1637, 1648, 1705, 1711, 1800, 1855, 2216, 2223, 2367, 2374, 2498, 2537, 2664, 2666, 2672, 2682, 2684, 2690, 2713, 2715, 2721, 2739, 2741, 2747, 2775, 2779, 2791, 2794, 4649, 4650, 4652, 4668, 4670, 4887, 4888, 4891, 4904, 4910, 5070, 5083, 5089, 5173, 5236, 5281, 5495, 6289, 6344, 6367, 6390, 6684, 6897, 6901	$\backslash c_false_bool$	2111, 2119, 2132, 2138, 2144, 3985, 4001, 4017
$\backslash arraystretch$	1174, 3585, 3603, 5795, 5905, 5922, 6873, 6876	$\backslash g_tmpa_bool$	4802, 4810, 4837, 4845, 4850, 4983, 4991, 5018, 5026, 5031, 5302, 5313, 5324, 5335
$\backslash AutoNiceMatrix$	6540	$\backslash g_tmpb_bool$	1701, 1731, 1870
$\backslash AutoNiceMatrixWithDelims$	6492, 6532, 6544	$\backslash l_tmpb_bool$	5194, 5203, 5260, 5377, 5391, 5409, 5431, 5444
B		$\backslash c_true_bool$	2090
$\backslash baselineskip$	119, 125, 1992	box commands:	
		$\backslash box_clear_new:N$	1170, 1292
		$\backslash box_dp:N$	915, 935, 972, 992, 1047, 1198, 1207, 1282, 2331, 2388, 3603, 5877, 6876
		$\backslash box_gclear_new:N$	5784

<code>\box_grotate:Nn</code>	5844	<code>\cs_gset:Npn</code>	120, 126, 5493
<code>\box_ht:N</code>	916, 937, 943, 955, 978, 990, 1039, 1200, 1202, 1205, 1280, 1959, 1983, 1985, 1991, 2327, 2387, 3585, 5868, 6873	<code>\cs_gset_eq:NN</code>	78, 1191, 1507, 3143
<code>\box_ht_plus_dp:N</code>	2556, 2569	<code>\cs_if_exist:Ntf</code>	66, 1260, 1262, 1411, 1508, 1511, 1695, 2052, 3190, 3191, 3287, 3300, 3355, 3368, 3458, 3480, 3588, 3606, 4136, 4154, 4211, 4229, 5198, 5480, 5533, 6149, 6167, 6668, 6843, 6850, 6879, 6886, 7141, 7162
<code>\box_move_down:nn</code>	1047, 1989	<code>\cs_if_exist_p:N</code>	530, 4881, 5062, 5374, 5393, 5411
<code>\box_move_up:nn</code>	85, 87, 89, 1039, 2431, 2504, 2543	<code>\cs_if_free:Ntf</code>	74, 2982, 3500, 3539, 3620, 3695, 3743
<code>\box_rotate:Nn</code>	949	<code>\cs_if_free_p:N</code>	4282, 4284
<code>\box_set_dp:Nn</code>	971, 991, 2388	<code>\cs_new_protected:Npx</code>	3170, 4293
<code>\box_set_ht:Nn</code>	977, 989, 2387	<code>\cs_set:Nn</code>	688, 691, 694
<code>\box_set_wd:Nn</code>	965, 2438	<code>\cs_set:Npn</code>	116, 117, 122, 123, 128, 140, 141, 155, 157, 158, 164, 166, 188, 190, 194, 196, 372, 1141, 1483, 1484, 2238, 2269, 2984, 3246, 3308, 3376, 4178, 4253, 5161, 5162, 5168, 5169, 5265, 5795, 5905, 5922
<code>\box_use:N</code>	426, 956, 1046, 1994	<code>\cs_set_nopar:Npn</code>	1080, 1174, 1185, 5675, 5676
<code>\box_use_drop:N</code>	997, 1003, 1022, 2036, 2354, 2390, 2431, 2432, 2444, 2864, 5887, 6236, 6273	<code>\cs_set_nopar:Npx</code>	1081
<code>\box_wd:N</code>	427, 966, 994, 1001, 1060, 1303, 1305, 1566, 1574, 2439, 2441, 2563, 2575, 2859, 2862, 2904, 2908, 5856, 6558	<code>\cs_set_protected:Npn</code>	6529
<code>\l_tmpa_box</code>	409, 426, 427, 1302, 1303, 1304, 1305, 1629, 2387, 2388, 2390, 2431, 2432, 2556, 2569	<code>\cs_set_protected_nopar:Npn</code>	5772, 6104
<code>\l_tmpb_box</code>	2549, 2563, 2564, 2575		
C			
<code>\c</code>	68, 1723	D	
<code>\Cdots</code>	1220, 3996, 3999	<code>\Ddots</code>	1222, 4029, 4030, 4035, 4036
<code>\cdots</code>	1149, 1212	<code>\ddots</code>	1151, 1214
<code>\cellcolor</code>	1158, 1355	<code>\diagbox</code>	1234, 5772, 6104
<code>\centering</code>	1928, 5812	dim commands:	
char commands:			
<code>\char_set_catcode_space:n</code>	1679	<code>\dim_add:Nn</code>	5644
<code>\chessboardcolors</code>	1363	<code>\dim_compare:nNnTF</code> ..	119, 125, 963, 969, 975, 1756, 2697, 2908, 2972, 3468, 3490, 3679, 4366, 4377, 5557, 5567, 6159, 6179, 6558
<code>\cline</code>	176, 1217, 1218	<code>\dim_compare_p:n</code>	5804
clist commands:			
<code>\clist_clear:N</code>	350, 4683	<code>\dim_gzero:N</code>	967, 973, 979
<code>\clist_if_empty:Ntf</code>	4809, 4990, 6069	<code>\dim_max:nn</code>	5546, 5550
<code>\clist_if_empty_p:N</code>	2766	<code>\dim_min:nn</code>	5538, 5542
<code>\clist_if_in:NnTF</code>	348, 1125, 1708, 2220, 2371, 4962, 5157, 6411, 6413, 6415, 6417, 6436	<code>\dim_ratio:nn</code>	3734, 3782, 3900, 3905, 3916, 3924, 3933, 3938, 3949, 3957
<code>\clist_if_in:nnTF</code>	6397	<code>\dim_set:Nn</code>	5519, 5526, 5537, 5541, 5545, 5549, 5561, 5562, 5571, 5572, 5616, 5629
<code>\clist_map_inline:Nn</code>	351, 4630, 4653, 4684, 5340, 6395, 6921, 6938	<code>\dim_set_eq:NN</code>	5517, 5524, 5624, 5638
<code>\clist_map_inline:nn</code> ..	2957, 4493, 4542, 6456	<code>\dim_sub:Nn</code>	5641
<code>\clist_new:N</code>	309, 325, 326, 327, 328, 504	<code>\dim_use:N</code>	5558, 5568, 5619, 5620, 5632, 5633, 5656, 5657, 5658, 5659, 5683, 5684, 5685, 5686
<code>\clist_put_right:Nn</code>	360, 4701	<code>\dim_zero_new:N</code>	5516, 5518, 5523, 5525
<code>\clist_set:Nn</code>	601, 602, 606, 607, 636, 637, 638	<code>\c_max_dim</code>	3455, 3468, 3477, 3490, 5517, 5519, 5524, 5526, 5558, 5568, 6146, 6159, 6164, 6179, 6664, 6665, 6837, 6838, 6858, 6861, 7138, 7152, 7159, 7173
<code>\clist_set_eq:NN</code>	4682	<code>\dotfill</code>	1232, 6547
<code>\l_tmpa_clist</code>	350, 360, 362, 4682, 4684	<code>\dots</code>	1153
<code>\CodeAfter</code>	883, 1233, 2601, 2604, 2832, 2877, 3124, 7708	<code>\doublerulesep</code>	1856, 4891, 4905, 5070, 5084, 5174, 5237, 5282
<code>\CodeBefore</code>	1487	<code>\doublerulesepcolor</code>	122
<code>\color</code>	120, 126, 188, 190, 194, 196, 893, 1633, 1651, 3505, 3508, 3511, 3544, 3547, 3550, 3625, 3628, 3632, 3700, 3748, 4168, 4171, 4174, 4243, 4246, 4249, 4266, 4391, 4430, 5791, 5955, 6693, 7056, 7080	<code>\downbracefill</code>	7210
<code>\colorlet</code> ...	257, 258, 900, 907, 1164, 2847, 2891	<code>\draw</code>	3813, 4943, 5138
<code>\columncolor</code>	1160, 1362, 3133, 4733		
<code>\cr</code>	152, 178, 2826, 7206, 7210, 7240, 7242	E	
<code>\crr</code>	2637, 7205, 7237	<code>\egroup</code>	1484, 1677
cs commands:			
<code>\cs_generate_variant:Nn</code>	67, 180, 3825, 4422, 4423, 5500, 5501	<code>\else</code>	2238

else commands:

<code>\else:</code>	251, 4401
<code>\endarray</code>	1482, 1484, 2589, 2617
<code>\endBNiceMatrix</code>	6479
<code>\endbNiceMatrix</code>	6476
<code>\endgroup</code>	2246
<code>\endNiceArray</code>	2979, 3000, 3009
<code>\endNiceArrayWithDelims</code>	2926, 2936
<code>\endpgfscope</code>	3870, 6611
<code>\endpNiceMatrix</code>	6467
<code>\endsavenotes</code>	1686
<code>\endtabular</code>	1484
<code>\endtabularnotes</code>	2472
<code>\endVNiceMatrix</code>	6473
<code>\endvNiceMatrix</code>	6470
<code>\enskip</code>	2081, 2109, 2117, 2130, 2136
<code>\ensuremath</code>	3990, 4006, 4022, 4046, 4069
<code>\everycr</code>	151, 178, 1195, 7202, 7232
<code>\everypar</code>	1957, 1960

exp commands:

<code>\exp_after:wN</code>	214, 1540, 1634, 1652, 1716, 2243, 6903
<code>\exp_args:NNc</code>	5482
<code>\exp_args:Nne</code>	2964
<code>\exp_args:NNV</code>	2609, 3977, 3993, 4009, 4025, 4049, 4107, 4184, 4262, 6792
<code>\exp_args:NnV</code>	5217, 5240
<code>\exp_args:NNx</code>	1124, 2219, 2370
<code>\exp_args:Nnx</code>	2169, 5221
<code>\exp_args:No</code>	3801
<code>\exp_args:NV</code>	1696, 2244, 2585, 2612, 2614, 4942, 5137, 6298
<code>\exp_args:Nxx</code>	5765, 5766
<code>\exp_last_unbraced:NV</code>	1372, 3126, 6081
<code>\exp_not:N</code>	51, 52, 55, 56, 1800, 1849, 1921, 1923, 1928, 1929, 1930, 3021, 3035, 3043, 3045, 3136, 4370, 4381, 4391, 4733, 4943, 5138, 5287, 5359, 5822, 5833, 5912, 5930, 6084, 6502, 7009
<code>\exp_not:n</code>	1071, 1683, 3137, 4117, 4118, 4194, 4710, 4721, 4723, 4734, 5224, 5781, 5885, 5897, 5898, 6036, 6046, 6062, 6074, 6086, 6113, 6504, 6568, 6569
<code>\expandafter</code>	2983
<code>\ExplSyntaxOff</code>	76, 1685, 5497
<code>\ExplSyntaxOn</code>	73, 1678, 5490
<code>\extrarowheight</code>	3585, 5796, 5906, 5923, 6873

F

<code>\fi</code>	130, 1698, 2238, 2241, 4406, 5161, 5184
fi commands:	
<code>\fi:</code>	253, 4403
<code>\five</code>	3195, 3200
flag commands:	
<code>\flag_clear_new:n</code>	7002
<code>\flag_height:n</code>	7027
<code>\flag_raise:n</code>	7026
<code>\fontdimen</code>	2427
fp commands:	
<code>\fp_eval:n</code>	3841
<code>\fp_to_dim:n</code>	3876
<code>\futurelet</code>	135

G

<code>\globaldefs</code>	1673, 6015
group commands:	
<code>\group_insert_after:N</code>	6552, 6553, 6555, 6556

H

<code>\halign</code>	1209, 7203, 7235
<code>\hbox</code>	1088, 1638, 2442, 2504, 2543, 2662, 2680, 2707, 2711, 2737, 2773
hbox commands:	
<code>\hbox:n</code>	85, 87, 90
<code>\hbox_gset:Nn</code>	5786
<code>\hbox_overlap_left:n</code>	1040, 1048, 2641, 2860
<code>\hbox_overlap_right:n</code>	426, 2805, 2906
<code>\hbox_set:Nn</code>	409, 1037, 1302, 1304, 1629, 1987, 2187, 2549, 2564, 6116
<code>\hbox_set:Nw</code>	887, 1307, 2025, 2343, 2835, 2881
<code>\hbox_set_end:</code>	985, 1550, 2033, 2351, 2854, 2899
<code>\hbox_to_wd:nn</code>	451, 476, 7208, 7238
<code>\Hdotsfor</code>	1227, 7341, 7542
<code>\hdotsfor</code>	1154
<code>\hdottedline</code>	1224
<code>\heavyrulewidth</code>	2482
<code>\hfil</code>	1779, 2293, 7205, 7237
<code>\hfill</code>	148, 173
<code>\Hline</code>	1225, 5164
<code>\hline</code>	128
hook commands:	
<code>\hook_gput_code:nnn</code>	23, 31, 94, 110, 199, 205, 373, 487, 490, 493, 506, 700, 716, 1240, 3168, 3973, 4103, 4180, 4258, 4291, 6788
<code>\hrule</code>	132, 148, 173, 1133, 2482, 6698, 7061, 7085
<code>\hsize</code>	1971
<code>\hskip</code>	131
<code>\Hspace</code>	1226
<code>\hspace</code>	4082
<code>\hss</code>	1779, 2293

I

<code>\ialign</code>	1085, 1185, 1208, 3117, 5994
<code>\iddots</code>	1223, 4053, 4054, 4059, 4060
<code>\iddots</code>	80, 1152, 1215
if commands:	
<code>\if_mode_math:</code>	251, 4397
<code>\IfBooleanTF</code>	1541
<code>\ifnum</code>	130, 4362, 5161, 5184
<code>\ifstandalone</code>	1511
<code>\ignorespaces</code>	2276, 4408
int commands:	
<code>\int_case:nnTF</code>	4027, 4033, 4051, 4057
<code>\int_compare:nNnTF</code>	143, 144, 168, 885, 886, 897, 904, 940, 950, 1130, 1132, 1264, 1270, 1275, 1553, 1556, 1582, 1586, 1597, 1601, 1602, 1669, 1982, 2167, 2247, 2274, 2460, 2499, 2538, 2610, 2638, 2758, 2884, 3262, 3269, 3273, 3275, 3330, 3337, 3341, 3343, 3507, 3546, 3627, 3662, 3664, 4170, 4245, 4349, 4599, 4644, 4662, 4698, 4729, 4746, 4747, 4754, 4755, 4790, 4812, 4816, 4824, 4993, 4997, 5005, 5104, 5114, 5437, 5439, 5441, 5443, 5790, 5792, 5849, 5861, 6207, 6239, 6243, 6316, 6318, 6506, 6508, 6510, 6514, 6516, 6518, 6520, 6522, 6907, 6909

`\pgfpathmoveto` 4900, 4906,
 5079, 5085, 6429, 6442, 6589, 6913, 6930, 6956
`\pgfpathrectanglecorners` 4672, 4894, 5073, 6324
`\pgfpointadd` 467
`\pgfpointanchor` 197, 226, 3433,
 3444, 3461, 3483, 3591, 3609, 5536, 5544,
 6154, 6172, 6192, 6194, 6211, 6248, 6671,
 6846, 6853, 6882, 6889, 7001, 7005, 7144, 7165
`\pgfpointdiff` .. 468, 1335, 1341, 1408, 1422, 1423
`\pgfpointlineattime` 3835
`\pgfpointorigin` 2647, 2816, 3201
`\pgfpointscale` 467
`\pgfpointshapeborder` 4303, 4306
`\pgfrememberpicturepositiononpagetrue` ...
 918, 1013,
 1100, 1445, 2646, 2670, 2688, 2719, 2745,
 2785, 2814, 3179, 3206, 3790, 4302, 4870,
 4918, 4934, 5051, 5097, 5129, 5579, 5589,
 5600, 6119, 6292, 6407, 6584, 6657, 6834, 7134
`\pgfscope` 3832, 6601
`\pgfset` 436,
 461, 1014, 6225, 6262, 6600, 6678, 6836, 7178
`\pgfsetbaseline` 1012
`\pgfsetcornersarced` 4671, 6300
`\pgfsetlinewidth` .. 4910, 5089, 6327, 6410, 6901
`\pgfsetrectcap` 4911, 5090
`\pgfsetroundcap` 6597
`\pgfsetstrokecolor` 6298
`\pgfsyspdfmark` 74, 75
`\pgftransformrotate` 3839
`\pgftransformshift` 442, 467, 3217, 3833, 6224,
 6246, 6602, 6612, 6680, 6979, 6989, 7189, 7221
`\pgfusepath` 3859, 3869, 4433, 4897, 6328
`\pgfusepathqfill` 3971, 5076
`\pgfusepathqstroke`
 4912, 5091, 6431, 6444, 6598, 6915, 6932, 6966
`\phantom` 3990, 4006, 4022, 4046, 4069
`\pNiceMatrix` 6466
prg commands:
`\prg_do_nothing:`
 78, 208, 542, 1191, 3124, 4764, 4765
`\prg_new_conditional:Nnn` 4602, 4610
`\prg_replicate:nn` 415, 2728,
 2729, 4121, 4902, 5081, 6509, 6512, 6515, 6521
`\prg_return_false:` 4607, 4619
`\prg_return_true:` 4608, 4618
`\ProcessKeysOptions` 7277
`\ProvideDocumentCommand` 80
`\ProvidesExplPackage` 4

Q

`\quad` 396
quark commands:
`\q_stop` 140, 141, 157, 158, 161, 162,
 164, 165, 166, 187, 189, 193, 195, 341, 354,
 1372, 1394, 1540, 1716, 1789, 1870, 2104,
 2126, 2243, 2294, 2601, 2604, 4256, 4270,
 4271, 4481, 4486, 4546, 4634, 4635, 4657,
 4658, 4688, 4689, 4774, 4776, 5673, 5680,
 5722, 5723, 5727, 6081, 6283, 6306, 6315,
 6346, 6349, 6369, 6372, 6400, 6403, 6481,
 6496, 6776, 6781, 6807, 6814, 6903, 7012, 7020

R

`\raggedleft` 1930, 5813, 6616
`\raggedright` 1929, 1971, 5814
`\rectanglecolor` 1356, 3132
`\refstepcounter` 424
regex commands:
`\regex_const:Nn` 68
`\regex_match:nnTF` 6754
`\regex_replace_all:NnN` 1721
`\relax` 181, 182
`\renewcommand` 212
`\RenewDocumentEnvironment`
 6465, 6468, 6471, 6474, 6477
`\RequirePackage` 1, 3, 9, 10, 11
`\right` 1652, 2559, 2571, 6703, 7065, 7089
`\rotate` 1230
`\roundedrectanglecolor` 1357, 6084
`\rowcolor` 1159, 1358
`\rowcolors` 1359
`\rowlistcolors` 1360
`\RowStyle` 1236, 7760

S

`\savedanchor` 3195
`\savenotes` 1491
scan commands:
`\scan_stop:` 3127
`\scriptstyle`
 891, 2837, 2883, 3820, 3821, 3855, 3865
seq commands:
`\seq_clear:N` 1714
`\seq_clear_new:N` 4527, 5339
`\seq_count:N` 2611, 4418, 4572
`\seq_gclear:N` 1242, 1243, 1285,
 1286, 1288, 1517, 1518, 1519, 1520, 2487, 3125
`\seq_gclear_new:N` 1289, 1290, 1368, 2607, 2623
`\seq_gpop_left:NN` 2613, 2625
`\seq_gput_left:Nn` 667, 2249, 2251, 6054
`\seq_gput_right:Nn` 406, 1797, 2252,
 3387, 4417, 5882, 5894, 6066, 6571, 6760, 6782
`\seq_gset_from_clist:Nn`
 3021, 3035, 3043, 3045
`\seq_gset_map_x:NNn` 3149
`\seq_gset_split:Nnn` 2609, 2624
`\seq_if_empty:NTF` 2446, 3031, 3039, 5355, 6092
`\seq_if_empty_p:N` 4624
`\seq_if_in:NnTF`
 665, 4664, 4834, 4840, 4847, 5015,
 5021, 5028, 5539, 5547, 6152, 6170, 6756, 7327
`\seq_item:Nn`
 1268, 1273, 1323, 1324, 1325, 1326, 4591, 4593
`\seq_map_function:NN` 2615
`\seq_map_indexed_inline:Nn` 4413, 4428
`\seq_map_inline:Nn`
 1446, 2465, 2471, 2627, 3402, 4561, 4803,
 4805, 4807, 4984, 4986, 4988, 5432, 5995, 6905
`\seq_mapthread_function:NNN` 5668
`\seq_new:N` 240, 256, 291, 292, 293,
 294, 295, 296, 298, 299, 304, 308, 366, 7317
`\seq_put_right:Nn` 5419, 5946
`\seq_set_eq:NN` 4536
`\seq_set_filter:NNn` 4537, 4559
`\seq_set_from_clist:Nn` 5359, 7318
`\seq_set_map_x:NNn` 7323

<code>\xarraycr</code>	1475	<code>\tl_gclear:N</code>	882, 1522, 1529, 1700, 2242, 3123, 3128, 4432
<code>\xarraycr@array</code>	1475	<code>\tl_gclear_new:N</code>	1250, 1251, 1252, 1253, 1254, 1255, 1256, 1521
<code>\xhline</code>	136	<code>\tl_gput_left:Nn</code>	1065, 1733, 4731
<code>\array@array</code>	1478	<code>\tl_gput_right:Nn</code>	1065, 1558, 1745, 1799, 1847, 1861, 2089, 2110, 2118, 2131, 2137, 2143, 2204, 3019, 3033, 3041, 3134, 4109, 4186, 4342, 4351, 4363, 4368, 4379, 4390, 4420, 4708, 4719, 5176, 5270, 5285, 5290, 5357, 5456, 5774, 6033, 6043, 6059, 6071, 6082, 6094, 6106, 6561
<code>\bBigg@</code>	1302, 1304	<code>\tl_gset:Nn</code>	1493, 1494, 1495, 1682, 1704, 1710, 2076, 2077, 2107, 2133, 3136, 4418
<code>\c@MaxMatrixCols</code>	2951, 7360	<code>\tl_if_blank:nTF</code>	4445, 4448, 4454, 4457, 4463, 4466, 4472, 4475, 4579, 5721
<code>\c@tabularnote</code>	2449, 2460, 2488	<code>\tl_if_blank_p:n</code>	4637, 4641, 4691, 4695, 4882, 5063, 5731, 5736
<code>\col@sep</code>	1077, 1078, 1604, 1665, 2651, 2704, 2770, 2870, 2905, 3472, 3494	<code>\tl_if_empty:NnTF</code>	1120, 1530, 1539, 1632, 1650, 1815, 2459, 3103, 3104, 3130, 4340, 4388, 4576, 4659, 4660, 4862, 5043, 5200, 5789, 6057, 6079, 6294, 6692, 6902, 7055, 7079
<code>\CT@arc</code>	116, 117	<code>\tl_if_empty:nTF</code>	160, 645, 785, 820, 836, 852, 873, 1452, 2593, 2620, 3511, 3550, 3631, 3700, 3748, 4174, 4249, 4266, 6751, 7022, 7090, 7340
<code>\CT@arc@</code>	115, 120, 134, 147, 172, 188, 190, 1507, 2482, 3143, 4909, 4927, 5088, 5122, 6297, 6409, 6596, 6904	<code>\tl_if_empty_p:N</code>	1739, 1751, 3829, 3830
<code>\CT@drs</code>	122, 123	<code>\tl_if_empty_p:n</code>	2450, 5762
<code>\CT@drsc@</code>	126, 194, 196, 4881, 4882, 4886, 5062, 5063, 5067	<code>\tl_if_eq:NnTF</code>	1122, 1702, 2213, 2364, 2389, 2508, 4961, 5109, 5119, 5156, 6808, 6811, 6815, 6818, 6919, 6936
<code>\CT@everycr</code>	1189	<code>\tl_if_eq:nnTF</code>	657, 794, 2104, 2126, 4414
<code>\CT@row@color</code>	1191	<code>\tl_if_eq_p:NN</code>	3793
<code>\endarray@array</code>	1482	<code>\tl_if_exist:NnTF</code>	1524
<code>\if@firstamp</code>	2238	<code>\tl_if_in:NnTF</code>	4545, 4633, 4656, 4687, 5217
<code>\if@tempwa</code>	1698, 2241	<code>\tl_if_in:nnTF</code>	353, 2091, 2101, 2116
<code>\insert@column</code>	1473	<code>\tl_if_single_token:nTF</code>	594, 802
<code>\insert@column@array</code>	1473	<code>\tl_if_single_token_p:n</code>	67
<code>\NC@</code>	1141, 2984	<code>\tl_map_inline:nn</code>	2594
<code>\NC@do</code>	2983	<code>\tl_new:N</code>	248, 255, 267, 268, 269, 272, 279, 281, 306, 307, 311, 316, 367, 495, 499, 520, 522, 523
<code>\NC@find</code>	216	<code>\tl_put_left:Nn</code>	1169, 1377
<code>\NC@find@V</code>	1695	<code>\tl_put_right:Nn</code>	647, 1179, 1277, 1317, 1533
<code>\NC@list</code>	1698, 2241, 2983	<code>\tl_range:nnn</code>	105
<code>\NC@rewrite@S</code>	212	<code>\tl_set:Nn</code>	4487, 4488, 4547, 4563, 4566, 4643, 4645, 4661, 4663, 4697, 4699, 4798, 4979, 5748, 6317, 6319, 6350, 6351, 6373, 6374, 6404, 6405, 6809, 6812, 6816, 6819
<code>\new@ifnextchar</code>	1247	<code>\tl_set_eq:NN</code>	362, 497, 4484, 4485, 4646, 6347, 6348, 6370, 6371, 6401, 6402, 6779, 6780, 6810, 6813, 6817, 6820
<code>\newcol@</code>	1143, 1144, 2986, 2987	<code>\tl_set_rescan:Nnn</code>	2608, 3976, 4106, 4183, 4261, 6791
<code>\nicematrix@redefine@check@rerun</code>	97, 100	<code>\tl_to_str:n</code>	7324
<code>\pgf@relevantforpicturesizefalse</code>	1330, 1444, 3180, 3791, 3962, 4427, 4541, 4871, 4919, 4935, 5052, 5098, 5130, 5580, 5590, 5601, 6120, 6293, 6408, 6583, 6658, 6835, 7135	token commands:	
<code>\pgfsys@getposition</code>	1328, 1333, 1339, 1406, 1414, 1417	<code>\token_to_str:N</code>	7341, 7431, 7436, 7442, 7460, 7468, 7474, 7478, 7518, 7524, 7542, 7555, 7561, 7610, 7617, 7630, 7660, 7676, 7693, 7694, 7707, 7716, 7722, 7760, 7777, 7983
<code>\pgfsys@markposition</code>	1042, 1050, 1095, 1181, 1327, 2644, 2665, 2683, 2714, 2740, 2778, 2810		
<code>\pgfutil@check@rerun</code>	102, 103		
<code>\reserved@a</code>	135		
<code>\rvtx@ifformat@geq</code>	66		
<code>\set@color</code>	5790, 6116		
<code>\tikz@library@external@loaded</code>	1508		
tex commands:			
<code>\tex_mkern:D</code>	84, 86, 88, 91		
<code>\tex_the:D</code>	215		
<code>\textfont</code>	2427		
<code>\textit</code>	369		
<code>\textsuperscript</code>	370, 371		
<code>\the</code>	181, 182, 1698, 1716, 2241, 2243, 2983		
<code>\thetabularnote</code>	372		
<code>\tikzexternaldisable</code>	1510		
<code>\tikzset</code>	1349, 1512, 3107, 4942, 5137		
tl commands:			
<code>\tl_clear:N</code>	5192, 6288		
<code>\tl_clear_new:N</code>	4482, 4483, 4529, 4565, 6777, 6778, 6803, 6804, 6805, 6806		
<code>\tl_const:Nn</code>	51, 52, 55, 56, 496, 2828, 2873, 7013		
<code>\tl_count:n</code>	2408, 2520		
		U	
		<code>\UnderBrace</code>	3119, 7128, 7694
		<code>\unskip</code>	2475

<code>\upbracefill</code>	7240	<code>\Vdots</code>	1221, 4012, 4015
use commands:		<code>\vdots</code>	1150, 1213
<code>\use:N</code>	1068,	<code>\Vdotsfor</code>	1228
1312, 1313, 1527, 1545, 1546, 2946, 2966, 4431		<code>\vfill</code>	450, 476
<code>\use:n</code>	1914, 4267, 4738, 4943, 5138,	<code>\vline</code>	134
5820, 5831, 5910, 5928, 6354, 6377, 6500, 7008		<code>\VNiceMatrix</code>	6472
<code>\usepackage</code>	7303, 7313	<code>\vNiceMatrix</code>	6469
<code>\usepgfmodule</code>	2	<code>\vrule</code>	132, 1959, 2327, 2331
		<code>\vskip</code>	131
		<code>\vtop</code>	1092
	V		
vbox commands:			X
<code>\vbox:n</code>	90, 7233		
<code>\vbox_set_top:Nn</code>	952	<code>\xglobal</code>	900, 907, 1164, 2847, 2891
<code>\vbox_to_ht:nn</code>	447, 474, 2555, 2568		Z
<code>\vbox_to_zero:n</code>	954		
<code>\vbox_top:n</code>	7199		
<code>\vcenter</code>	1635, 2553, 6695, 7058, 7082, 7660	<code>\Z</code>	6754

Contents

1	The environments of this package	2
2	The vertical space between the rows	2
3	The vertical position of the arrays	3
4	The blocks	4
4.1	General case	4
4.2	The mono-column blocks	6
4.3	The mono-row blocks	6
4.4	The mono-cell blocks	6
4.5	Horizontal position of the content of the block	7
5	The rules	7
5.1	Some differences with the classical environments	8
5.1.1	The vertical rules	8
5.1.2	The command <code>\cline</code>	8
5.2	The thickness and the color of the rules	9
5.3	The tools of nicematrix for the rules	9
5.3.1	The keys <code>hlines</code> and <code>vlines</code>	9
5.3.2	The keys <code>hvlines</code> and <code>hvlines-except-borders</code>	10
5.3.3	The (empty) corners	10
5.4	The command <code>\diagbox</code>	11
5.5	Dotted rules	11
5.6	Commands for customized rules	12
6	The color of the rows and columns	13
6.1	Use of <code>colortbl</code>	13
6.2	The tools of nicematrix in the <code>\CodeBefore</code>	14
6.3	Color tools with the syntax of <code>colortbl</code>	18
7	The command <code>\RowStyle</code>	18
8	The width of the columns	19
8.1	Basic tools	19
8.2	The columns V of <code>varwidth</code>	20
8.3	The columns X	21

9	The exterior rows and columns	21
10	The continuous dotted lines	23
10.1	The option nullify-dots	24
10.2	The commands <code>\Hdotsfor</code> and <code>\Vdotsfor</code>	25
10.3	How to generate the continuous dotted lines transparently	26
10.4	The labels of the dotted lines	27
10.5	Customisation of the dotted lines	27
10.6	The dotted lines and the rules	28
11	The <code>\CodeAfter</code>	28
11.1	The command <code>\line</code> in the <code>\CodeAfter</code>	28
11.2	The command <code>\SubMatrix</code> in the <code>\CodeAfter</code>	29
11.3	The commands <code>\OverBrace</code> and <code>\UnderBrace</code> in the <code>\CodeAfter</code>	31
12	The notes in the tabulars	32
12.1	The footnotes	32
12.2	The notes of tabular	32
12.3	Customisation of the tabular notes	34
12.4	Use of <code>{NiceTabular}</code> with <code>threeparttable</code>	36
13	Other features	36
13.1	Use of the column type S of <code>siunitx</code>	36
13.2	Alignment option in <code>{NiceMatrix}</code>	36
13.3	The command <code>\rotate</code>	36
13.4	The option <code>small</code>	37
13.5	The counters <code>iRow</code> and <code>jCol</code>	38
13.6	The option <code>light-syntax</code>	38
13.7	Color of the delimiters	39
13.8	The environment <code>{NiceArrayWithDelims}</code>	39
14	Use of Tikz with <code>nicematrix</code>	39
14.1	The nodes corresponding to the contents of the cells	39
14.1.1	The columns V of <code>varwidth</code>	40
14.2	The medium nodes and the large nodes	40
14.3	The nodes which indicate the position of the rules	42
14.4	The nodes corresponding to the command <code>\SubMatrix</code>	43
15	API for the developpers	44
16	Technical remarks	45
16.1	Definition of new column types	45
16.2	Diagonal lines	45
16.3	The empty cells	46
16.4	The option <code>exterior-arraycolsep</code>	46
16.5	Incompatibilities	47
17	Examples	47
17.1	Utilisation of the key 'tikz' of the command <code>\Block</code>	47
17.2	Notes in the tabulars	48
17.3	Dotted lines	49
17.4	Dotted lines which are no longer dotted	50
17.5	Stacks of matrices	51
17.6	How to highlight cells of a matrix	54
17.7	Utilisation of <code>\SubMatrix</code> in the <code>\CodeBefore</code>	56
18	Implementation	57
19	History	235

